The University of Alberta
Printing Department
Edmonton, Alberta

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: LLOYD DAVID BENBOW

TITLE OF THESIS: On the Structure and Use of Hybrid Sequential Machines

DEGREE FOR WHICH THESIS WAS PRESENTED: M. Sc.

YEAR THIS DEGREE GRANTED: 1974

THE UNIVERSITY OF ALBERTA

On the Structure and Use of Hybrid Sequential Machines

by

LLOYD DAVID BENBOW

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

AND RESEARCH IN PARTIAL FULFILMENT

OF THE REQUIREMENTS FOR THE

DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1974

THE UNVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled "On the Structure and Use of Hybrid Sequential Machines", submitted by Lloyd David Benbow in partial fulfilment of the requirements for the degree of Master of Science.

ABSTRACT

This thesis is concerned with the problems of combined asynchronous and synchronous machines which are called hybrid machines. Initially conditions are given for connecting an asychronous machine to a synchronous machine and vice versa. Methods are presented for decomposing, if it is possible, a given sequential machine into two machines, one which is synchronous and the other asynchronous. A theorem for inserting delay elements into asynchronous circuits to form a hybrid circuit is presented. It is shown that hybrid machines can be used to eliminate not only critical races, but also essential hazards from asynchronous circuits.

ACKNOWLEDGMENTS

The author wishes to extend special thanks to his supervisor, Dr. W. A. Davis, for his guidance and assistance in this research.

Financial support from the Department of Computing Science, in the form of Graduate Teaching Assistantships, and a Graduate Research Assistantship, is gratefully acknowledged. The support extended by the National Research Council under Grant A7634 is appreciated.

The help and encouragement offered by fellow students, especially Ernst J. Schuegraf, is gratefully acknowledged. Finally, the author expresses appreciation to Jean Wilson for her assistance in proof reading the thesis.

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF TABLES

LIST OF TABLES (continued)

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This thesis is primarily concerned with the relationship between two classes of sequential machines: synchronous and asynchronous. In brief the main object is to combine the two classes of machines. Different approaches are taken: One method is by studying the connection of machines. Another is by examining a combined asynchronous-synchronous machine, which is called a hybrid machine. In the study of hybrid machines the possibility of replacing an asynchronous or synchronous circuit with a hybrid circuit is examined. In addition, the usefulness of such a replacement is discussed.

Each class of sequential machines has its advantages and depending on the situation, one type of machine is usually more suitable than the other. In some circumstances, however, a hybrid machine may be better than either an asynchronous or synchronous machine since the advantages of both asynchronous and synchronous machines can be utilized. With the advent of integrated circuits the

practical aspects of sequential circuits has changed from connecting individual gates together to interconnecting complete circuits. These components can in fact be small machines and the timing and connection problems that exist with gates merely increase as the complexity increases. The results presented here will be useful to circuit designers and are independent of the size of the circuit.

The problems of connecting sequential machines together are studied in Chapter 2. Two types of connections are considered: parallel and serial. In each case different types of circuits are discussed, that is, synchronous-synchronous, asynchronous-asynchronous and synchronous-asynchronous connections. The conditions and limitations imposed on each of the machines for the various types of connections are given.

The reverse approach is taken in Chapter 3, where machines are considered from the point of view of their decomposition into different types of machines. Methods for the decomposition into synchronous-synchronous, asynchronous-asynchronous, and asynchronous-synchronous machines are presented.

In Chapter 4, hybrid machines are developed. Conditions under which synchronous and asynchronous circuits can be converted into hybrid circuits are presented. It is also illustrated how a hybrid machine can be obtained from either asynchronous or synchronous circuits.

The problems and uses of hybrid machines are covered in Chapter 5. Solutions are given for the problem of eliminating races in hybrid machines. In addition, a discussion is presented how a hybrid machine can be used to resolve critical races and essential hazards in asynchronous circuits.

1.2 Fundamentals

A basic knowledge of switching theory is assumed; however, a brief summary is presented in order to establish the notation used.

A sequential circuit can be represented schematically by Fig. 1.1. This circuit has a finite number of inputs denoted by $x_1,\ldots,x_n$ and a finite set of outputs denoted by $z_1,\ldots,z_m$. The signal values at the outputs of all of the delay elements are referred to as the state variables, and are denoted by $y_1,\ldots,y_p$. The various combinations of values of the $y_1,..,y_p$ variables define the internal state of the machine. The combinational circuit produces the outputs $z_1,z_2,\ldots,z_m$ and the next-state variables $Y_1',\ldots,Y_p'$ from the inputs and the internal state variables. The next-state variables at time t are identical with the internal state variables at time $t+\Delta$. In other words, $Y_1',\ldots,Y_p'$ define the internal state that the circuit will assume next. The circuit also contains a delay element for each feedback path. D_k denotes the delay element of the k-th feedback

path. Besides these delays that are deliberately introduced by the designer, the circuit may contain stray delays. A stray delay is defined as a delay not under the control of the designer [32].

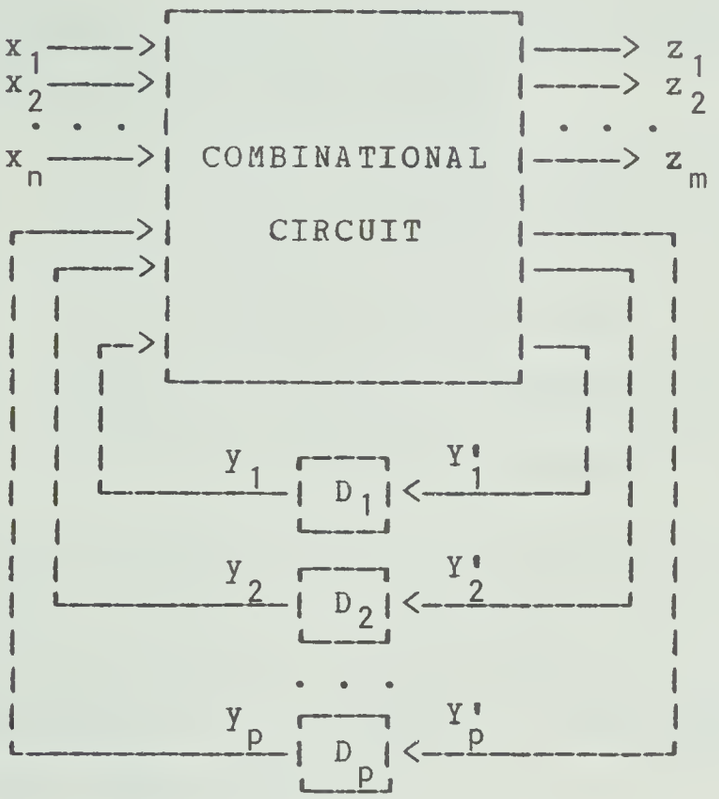


FIGURE 1.1: SCHEMATIC DIAGRAM OF A SEQUENTIAL CIRCUIT

The variables of Fig. 1.1 are all assumed to be binary, and therefore restricted to the values 0 and 1. The set $I=\{I_1,I_2,\ldots,I_k\}$ of possible configurations of n binary input variables $x_1,\ldots,x_n$ will define the inputs of a sequential machine. Likewise, the set $Z=\{Z_1,Z_2,\ldots,Z_\ell\}$ of m binary variables $z_1,\ldots,z_m$ defines the outputs. The combination of p internal variables $y_1,\ldots,y_p$ defines the

states of a sequential machine and will be denoted by S, $S=\{s_1,s_2,\ldots,s_r\}$. With the introduction of these sets of binary variables it is now possible to present the mathematical model of a sequential machine.

Definition 1.1 [12] A sequential machine is a quintuple $M=\{S,I,Z,N,O\}$ where

1. S is a finite nonempty set of states;
2. I is a finite nonempty set of inputs;
3. Z is a finite nonempty set of outputs;
4. $N:S\times I\rightarrow S$ is called the next-state function;
5. $O:S\times I\rightarrow Z$ is called the output function;

The cartesian product $S\times I$ is the set containing all pairs of elements (s_i,I_j). A pair (s_i,I_j) will be referred to as the total state of the machine. The total output state is the pair $(N(s_i,I_j),Z_k)$.

The two distinct classes of sequential machines are termed asynchronous and synchronous. A synchronous machine is a machine in which all internal variables change simultaneously. The behavior is governed by the values of its internal variables at definite instants of time, and is unaffected by their values at any other time. Synchronization is accomplished by a timing device called a clock, which simply generates a sequence of pulses which are input to all delay elements. These pulses merely regulate the delays in the feedback paths such that they all operate

simultaneously. It is assumed that each feedback path contains at least one clocked delay element.

On the other hand, an asynchronous machine is a machine whose behavior depends upon the order in which its internal variables change. In an asynchronous machine the change of its internal variables are not simultaneous. There is no clock pulse to control the delay elements in the feedback paths. In fact, some or all of the feedback paths may be free of delay elements.

Recently a new class of sequential machine, the hybrid, has been introduced [9]. A hybrid machine is a sequential circuit in which some of the internal variables change at specific instants of time, and the remaining internal variables change continuously. Expressed another way, the behavior is governed by the values of some, but not all, internal variables at well defined instants of time, and on the order in which the other variables change.

A sequential machine is said to have a fundamental next state table if for any total state (s_i, I_t) for which $N(s_i, I_t) = s_j$, then $N(s_j, I_t) = s_j$. If the sequential machine is asynchronous and has a fundamental next state table then the machine is said to operate in fundamental mode. An asynchronous machine is said to operate in normal mode if and only if the inputs are never changed unless the circuit is in a stable state.

CHAPTER 2

COMPOSITION

Investigations of sequential machines have usually been divided into two classes. That is, theories pertaining to the synchronous case have been handled separately from the theories pertaining to the asynchronous case. In order to bring the two theories closer together this thesis will first look at conditions under which two machines, of either class, can be connected together. This chapter will deal exclusively with compositions of only two machines; however, this can be easily extended to any number of machines.

The theories on composition have been stated mainly for the synchronous case [13]. In this chapter the basic theories already developed will be briefly discussed. In addition, the theories will be applied to the asynchronous case. Conditions will also be presented that assure the existence of a composition consisting of both types of machines.

Before going into details it is necessary to define the terminology that will be used. In serial composition the

first component will be referred to as the front machine. The second component will be called the tail machine. The tail machine is nontrivially dependent on the front machine, which means that some of the outputs of the front machine are inputs to the tail machine. When talking of synchronous machines it is necessary to define a period of time, say E, with reference to the clock pulse such that if a machine contains flip-flops that trigger on:

1. the leading edge, then the clock changes from 0 to 1 in time E.
2. the trailing edge, then the clock changes from 1 to 0 in time E.
3. the leading and trailing edge i.e., in the master-slave case then the clock pulse (0-1-0) is less than E.

For the purpose of this thesis it will be assumed that only one flip-flop will be present in each feedback path. Also, it is assumed that all flip-flops of a circuit are of the same type, or that they all trigger on the same edge of the clock pulse. This ensures that all delay elements will change simultaneously.

Composition is presented in two major sections: the first section will deal with parallel composition, the other with serial composition. In each case, conditions for the composition of different types of machines will be given. For the serial case, conditions will be discussed which are

necessary for the combination of a synchronous front machine with an asynchronous tail machine. In addition, conditions will be stated to allow the connection of an asynchronous front machine with a synchronous tail machine.

2.1 Parallel Composition

The parallel composition of two machines M^1 and M^2 as described by Fig. 2.1 has the following formal definition:

DEFINITION 2.1 [13] The general parallel composite machine corresponding to two machines

$M^1 = \{S^1, I^1, Z^1, N^1, O^1\}$

$M^2 = \{S^2, I^2, Z^2, N^2, O^2\}$ is the machine

$M^{12} = \{S^1 \times S^2, I^1 \times I^2, Z^1 \times Z^2, N^{12}, O^{12}\}$ such that

$N^{12}(s_i^1 s_j^2, x^1 x^2) = N^1(s_i^1, x^1), N^2(s_j^2, x^2)$ and

$O^{12}(s_i^1 s_j^2, x^1 x^2) = O^1(s_i^1, x^1), O^2(s_j^2, x^2)$ where

$s_i^1 \varepsilon S^1$ and $s_j^2 \varepsilon S^2$ and $x^1 \varepsilon I^1$ and $x^2 \varepsilon I^2$

Machines M^1 and M^2 in this definition are regarded as synchronous; however, under certain conditions M^1 or M^2 can be asynchronous. That is, M^1 and M^2 could be both synchronous, both asynchronous or one could be asynchronous with the other synchronous. The conditions are dependent not only on the individual machines, but also on the machines they will be connected with. In other words, the conditions that a synchronous machine can be connected to a synchronous machine are different from the conditions under

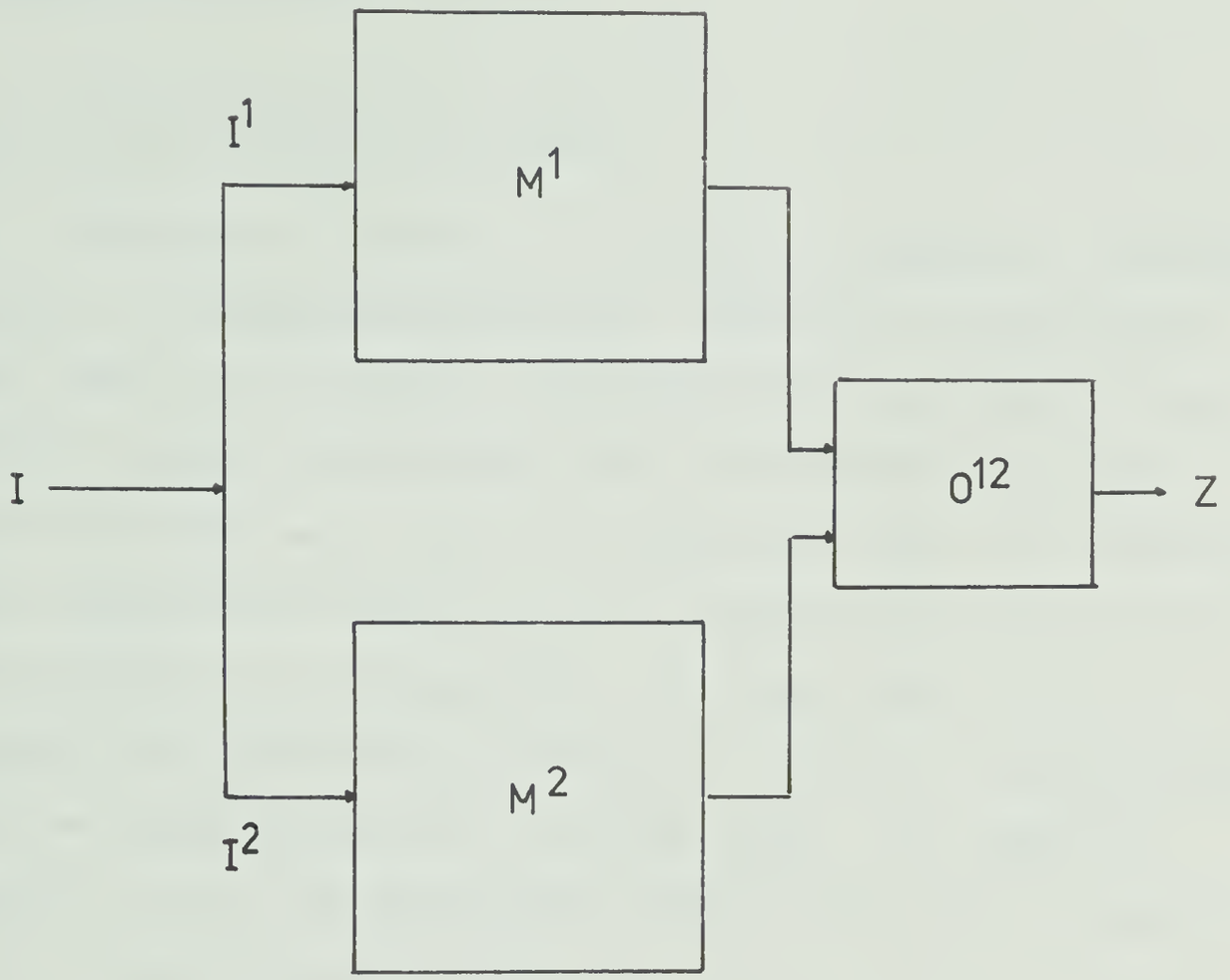


Figure 2.1

Schematic Diagram of Parallel Composition

which it can be connected to an asynchronous machine. Parallel composition is dealt with in different subsections, since the conditions are dependent on the type of machines connected together.

2.1.1 Two Synchronous Machines

The general definition of parallel composition pertains mainly to synchronous machines. Therefore the conditions that need to be placed on two synchronous machines may seem rather trivial, nevertheless they are briefly mentioned. The conditions which must be imposed are, that the inputs to both machines are stable for the duration of E and that the two machines are synchronized. In other words, it is assumed that both machines operate correctly. Provided that the two machines also use the same clock, the composite machine of any two synchronous machines will be the machine M^{12} as stated in Definition 2.1.

2.1.2 Two Asynchronous Machines

Two asynchronous machines can be connected using Definition 2.1 provided that: the machines are free of critical races, and there exist no steady-state hazards [31] in either circuit. Again it should be noted that these conditions need not be stated if it is assumed that both machines operate correctly. The resulting composite machine M^{12} will operate according to Definition 2.1 if the above

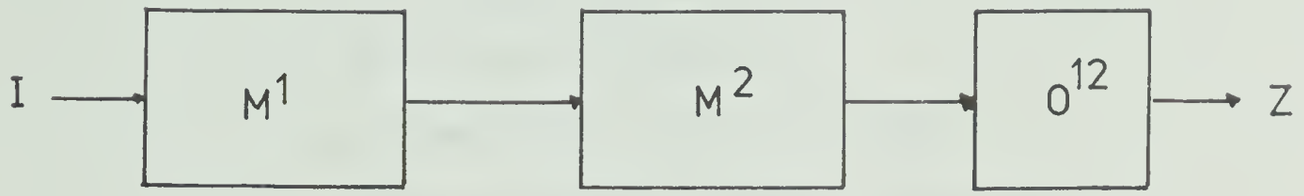
conditions are met. It should be noted that the output is not determined until both machines reach a stable state.

2.1.3 Asynchronous-Synchronous

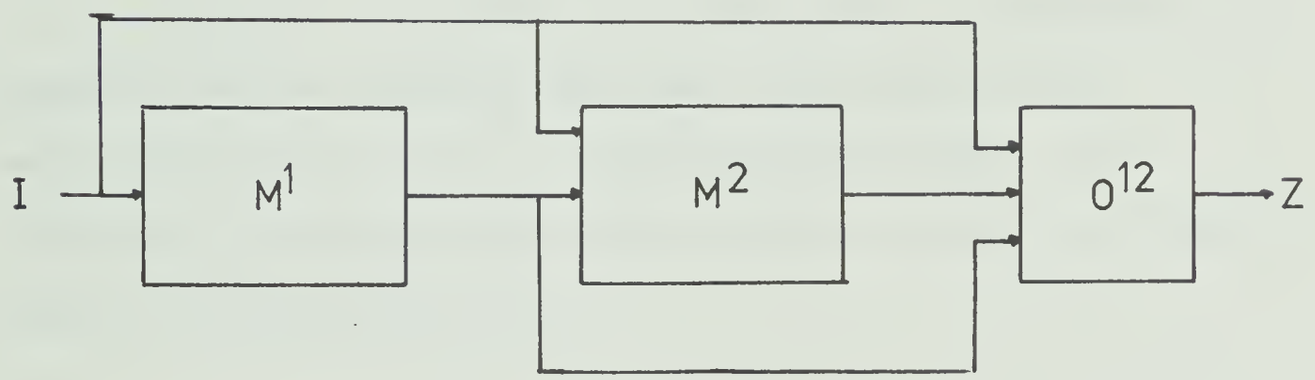
A composite machine consisting of an asynchronous machine and a synchronous machine will function according to the definition, provided that the inputs to the synchronous machine are stable during E, and the asynchronous machine operates correctly. This implies, that the asynchronous machine is free of critical races and steady-state hazards.

2.2 Serial Composition

Serially connected machines can operate in two different ways. One way is to have the outputs of the front machine as the inputs to the tail machine, as shown in Fig. 2.2(a). The other way is to use the inputs and internal variables of the front machine as inputs to the tail machine, as seen in Fig. 2.2(b). It is sufficient to consider only the second case since the first is just a special case of the second. The general definition is now presented.



(a)



(b)

Figure 2.2

(a) Serial Composition with Outputs of M^1 as Inputs of M^2

(b) Serial Composition with Internal Variables and Inputs of M^1 as Inputs to M^2

DEFINITION 2.2 [13] The general serial composite machine corresponding to two machines

$M^1 = \{S^1, I^1, Z^1, N^1, O^1\}$

$M^2 = \{S^2, I^2, Z^2, N^2, O^2\}$

is the machine M^{12} such that

$M^{12} = \{S^1 x S^2, I^1, Z^2, N^{12}, O^{12}\}$ where

$I^2 = S^1 x I^1$ and

$N^{12}(s^1_i s^2_j, x) = N^1(s^1_i, x), N^2(s^2_j, (s^1_i, x))$ and the output function O^{12} is a function of S^1, S^2 and I^1.

Definition 2.2 which was first given by Hartmanis [13], pertains to the synchronous case, but under certain conditions other types of machines can be used. The following subsections describe the way in which two machines can be serially connected.

2.2.1 Two Synchronous Machines

The conditions needed for two synchronous machines to be serially connected are trivial, since Definition 2.2 relates specifically to synchronous machines. The only conditions needed are that the inputs to the system must be stable during E, and that both machines use the same clock.

2.2.2 Two Asynchronous Machines

Two conditions must be met in order to serially connect two asynchronous machines. The conditions are that the two

machines must be free of critical races and steady-state hazards, and that both machines must change state concurrently. It is, however, very unlikely to have the two machines change states concurrently and a more realistic possibility would be for one of the machines to change before the other. That is, M^1 might recognize the change in its own internal variables before M^2 recognizes the change in M^1. On the other hand M^2 might see the I^1 input change before M^1 sees it.

The condition that the two machines must operate concurrently can be removed provided that the two machines are consistent in recognizing the input change. In other words, if M^2 sees the I^1 input change before M^1 in any one circumstance then M^2 must always see the I^1 input change first. It is always possible to restrict the order of change of the two asynchronous machines, and two methods to do so are now given. One possible solution is to restrict the composition so that M^2 is forced to change before M^1. That is, the next state of M^2 is determined by the present state of M^1 and the inputs. The other possible solution is to restrict the order of change so that the state of M^1 changes before the state of M^2 changes. In this case the state of M^2 is determined by the inputs and the next state of M^1.

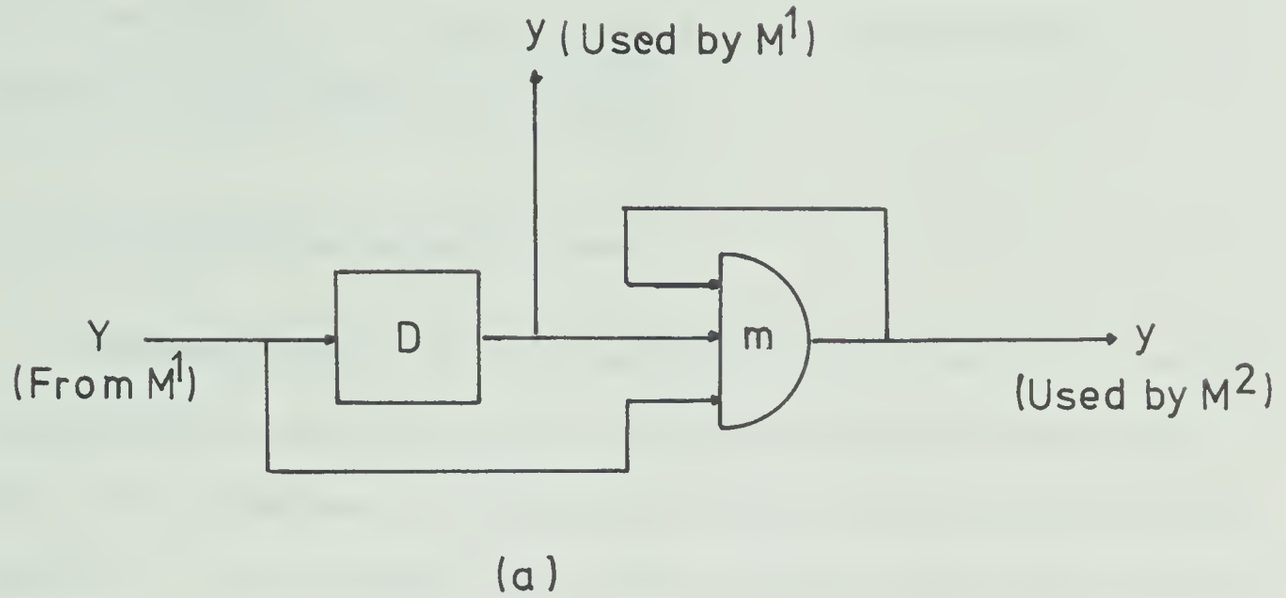
Both solutions can be implemented by inserting delay elements in the feedback paths of M^1. The inputs to the

tail machine are the outputs of the front machine, either before or after the delay elements, depending on the restrictions on the composition. It should be noted that if M^2 is forced to change before M^1 then the delay time in the feedback path of M^1 should be of sufficient magnitude. The magnitude of the delays should be such that M^2 reaches a stable state in response to the input change before the outputs of the delays in M^1 change. Also, the delays should be of a type where only the outputs of the stable states are used as inputs to M^2. This is easily accomplished by the circuit given in Fig. 2.3.

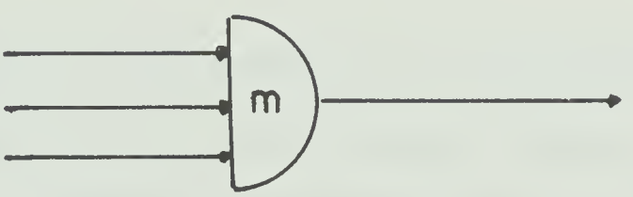
2.2.3 Asynchronous-Synchronous

In order for an asynchronous front and a synchronous tail machine to be connected serially, two conditions must be satisfied. These conditions are dependent on the type of inputs used by the tail machine. Again the asynchronous machine is restricted; it must be free of critical races and essential hazards.

The asynchronous front machine is required to reach a stable state before E and must remain stable during E. The reason is that the asynchronous y-variables and I^1 inputs are used as inputs to the synchronous machine. Also the inputs I^1 must be stable during E. It should be noted, that in reaching the stable state the order in which the asynchronous internal variables change is not important.



(a)



(b)

Figure 2.3

(a) Inertial Delay Device

(b) Majority Gate

If the tail machine is not directly dependent on the input I^1, then there is no need to have any restrictions on the inputs to the composite machine.

2.2.4 Synchronous-Asynchronous

The composition of two machines where the front machine is synchronous and the tail machine is asynchronous, is similarly dependent on the inputs used by the asynchronous machine. It is assumed that the inputs to the front machine are stable during E and that the tail machine is free of critical races and essential hazards. Also, imposed is the further restriction that I^1 must change sufficiently before the clock pulse.

If more than one of the y-variables of the synchronous machine are used as inputs to the asynchronous machine, then the tail machine will be required to handle multiple input changes. The reason is that the outputs and y-variables from the synchronous machine will change instantaneously with each clock pulse. It should be noted that the asynchronous machine will always react to the I^1 input change first, since the synchronous machine must wait for the clock pulse to change state. That is, the tail machine is forced to operate before the front machine. The composite machine will still operate correctly since this is one of the solutions stated in Section 2.2.2 for two asynchronous machines.

2.3 Summary

The attention in this chapter was focused on the two different types of machines, synchronous and asynchronous. In particular, two alternate ways of connecting two machines were studied. Parallel and serial connections have been presented for machines within one class, as well as for machines of different types. Furthermore, conditions were stated which allowed the connection of different classes of machines. The results developed above will be of value to the next chapter which deals with decomposition.

CHAPTER 3

DECOMPOSITION

The study of asynchronous and synchronous machines is continued in this chapter. It will examine methods of decomposing both types of machines. For each case, serial decomposition will be considered such that the submachines are synchronous, asynchronous or both. Parallel decomposition is dealt with in the last section, again with asynchronous and synchronous cases being considered.

Throughout the chapter a machine will be decomposed into two submachines only; however, the theories can be easily extended iteratively to any number of submachines. A machine M'' is said to be controlled by another machine M' if the machine M'' uses the outputs of the machine M' as inputs.

3.1 Synchronous Serial Decomposition

The algebra of partitions is used to develop theories and methods of decomposing synchronous machines. The basic terminology of partitions is summarized below, since they

are essential for an understanding of decomposition.

A partition on a set of states S is a grouping of states into disjoint subsets called blocks, such that every state belongs to exactly one block. The product $\pi \cdot \pi'$ of two partitions π and π' is obtained by intersecting the blocks of the individual partitions. A partition on a set of states S of a machine M is said to have the substitution property (SP) if and only if s_i and s_j are contained in the same block of π, then $N(s_i,x)$ is contained in the same block as $N(s_j,x)$ for all x in I.

A necessary and sufficient condition for the serial decomposition of a synchronous sequential machine is now stated.

> THEOREM 3.1 [13] A sequential machine M has a nontrivial serial decomposition of its next-state behavior if and only if there exists a nontrivial SP partition π on the set of states S of M.
>
> Proof: see [13].

The states of the two submachines of the decomposed machine are defined in the following manner. If π is an SP partition then the front machine has a state corresponding to each block of π. The states of the tail machine correspond to the blocks of some nontrivial partition π' such that $\pi \cdot \pi' = 0$. The two submachines obtained from the partitions π and π' will both be synchronous. It should be noted that if both partitions have SP this results in the

trivial case, i.e., the decomposition gives a parallel connection. For the remainder of this section it will be assumed that only one of the partitions has SP. Parallel decomposition will be dealt with in Section 3.3.

It is desirable to extend the theory of synchronous decomposition so that a synchronous machine can be decomposed into two submachines, one of which is asynchronous and one of which is synchronous. As stated previously there are two basic ways that two machines of different types can be connected. The asynchronous submachine can control the synchronous one, or the synchronous submachine can control the asynchronous one.

Most of the work done in the field of synchronous serial decomposition pertains to decomposing synchronous machines into two synchronous submachines[13]. The next two sections will study decomposition of synchronous machines into submachines of different types. Serial decomposition is presented such that a synchronous submachine controls an asynchronous submachine and vice versa.

3.1.1 Synchronous-Asynchronous Case

As explained earlier, a serial decomposition can be found, if there exist two partitions π and π' and if $\pi \bullet \pi' = 0$. The condition that either π or π' has SP on M must also be satisfied.

To obtain a serial decomposition for a synchronous machine controlling an asynchronous one, further conditions must be placed on one of the partitions. Clearly these conditions are placed on the partition used to realize the tail machine since it is asynchronous, and only certain partitions will realize a submachine of this type. The following theorem establishes the conditions which must be imposed on both partitions.

THEOREM 3.2 [7] A sequential machine M, given by its next-state behavior, can be decomposed into a nontrivial serial connection of a synchronous machine controlling an asynchronous machine if and only if there exist two nontrivial partitions π and π', on S, which satisfy the following conditions:

a. $\pi \cdot \pi' = 0$

b. π has SP on M

c. Let $\pi = \{A^1, A^2, \ldots, A^k\}$ and $\pi' = \{B^1, B^2, \ldots, B^\ell\}$.
If for any state a A^i, B^i and input x of M and $N(a,x) \in B^h$, and if $b \in B^h \cap A^i \neq \emptyset$ then $N(a,x) = N(b,x)$

Proof see [7].

Essentially all this theorem states is that the conditions for synchronous decomposition must exist and that the tail machine must have a fundamental next-state table. Conditions (a) and (b) guarantee synchronous serial decomposition. Condition (c) assures the existence of a

fundamental next-state table for the tail machine. Thus, if a partition π' can be found such that it gives a fundamental next-state table, then a decomposition having a front machine that is synchronous and an asynchronous tail machine is possible.

As an example of Theorem 3.2 consider the machine given in Table 3.1 and the two partitions $\pi = \{\overline{12};\overline{34};\overline{56}\}$ and $\pi' = \{\overline{135};\overline{246}\}$. Let A^1, A^2 and A^3 denote respectively the blocks $\overline{12}$, $\overline{34}$, $\overline{56}$ of π, and let B^1 and B^2 respectively denote the blocks $\overline{135}$ and $\overline{246}$ of π'. It can now be shown that the three conditions of Theorem 3.2 are satisfied by partitions π and π'. Conditions (a) and (b) are satisfied since $\pi \cdot \pi' = 0$ and π has SP on M. Condition (c) may be verified by looking at the following case. Considering state 1 and input 1 of Table 3.1 gives $1 \in A^1$, $N(1,1) \in B^2$ and $B^2 \cap A^1 = \overline{12} \cap \overline{246} = 2 \neq \emptyset$. Condition (c) is satisfied for this total state since $N(1,1) = N(2,1)$. Likewise, it can be shown that condition (c) is satisfied for all states and inputs for these two partitions. Therefore, a serial decomposition of a synchronous machine controlling an asynchronous machine exists for the machine defined by Table 3.1.

TABLE 3.1

State table of a synchronous machine

	0	1
1	3	6
2	4	6
3	5	1
4	6	1
5	2	3
6	2	4

This theorem gives no indication of how the two partitions π and π' might be found. The method for finding the partition π will not be described here, since π can be found in the same way as for the standard synchronous decomposition. However, a method for finding the partition π', so that it satisfies condition (c), is presented.

Before giving the rules for finding π' it is useful to introduce more notation. Let #(K) denote the number of states in a block $K \in \pi$, n(π) will denote the number of states in the largest block of the partition π. The statement identify a with b will mean that states a and b are placed in the same block of π, and will be denoted as $a \equiv b(\pi)$.

Let π be a patition with SP on M. Let a be any state of M belonging to block K of π such that #(K)=n(π). To find the corresponding conditions for π' the following rule is

used [7].

RULE 1 Let b be contained in the block K of π, such that $N(a,I)=N(b,I)$, for all I, then the following situations are possible:

1. No state b exists. Then to satisfy condition (c) of Theorem 3.2 for state a and input I, state a and $N(a,I)$ must be identified in π. However, observe that if $N(a,I)\varepsilon K$ the identification is possible only if $N(a,I)=a$, since $\pi\bullet\pi' =0$.
2. One state b exists. Then to satisfy condition (c) of theorem 3.2 for the state a and input I, two solutions are possible:
   1. identify a with $N(a,I)$ in π' or
   2. identify b with $N(a,I)$ in π

   If $N(a,I)\varepsilon K$ the second solution is possible only if $N(a,I)=b$.
3. m states b^i, i=1,2,...,m, exist. Then the following m+1 alternative solutions are possible.
   1. identify $N(a,I)$ with a in π' or
   2. identify $N(a,I)$ with b^1 in π' or
   3. identify $N(a,I)$ with b^2 in π' or

   .

   .

   .

   m+1. identify $N(a,I)$ with b^m in π'.

   If $N(a,I)\varepsilon K$ the i-th solution is possible only if $N(a,I)=b^i$.

Let $\#(K) < n(\pi)$ where K is mapped by I into a different block, then the following rule can be used to find the conditions placed on π'.

RULE 2 Place the $N'(a,I)$ in one of the $n(\pi) - \#(K)$ blocks that do not contain any state of M belonging to block K.

To obtain π', all the pairs (a,I) of M are examined and the corresponding conditions of π' are listed. If $\#(K) = n(\pi)$ where state a is contained in the block K, then the conditions are obtained by Rule 1 for all sets of input. If $\#(K) < n(\pi)$ the conditions are obtained by Rule 1 for inputs I mapping K into itself. For inputs mapping K into different blocks, Rule 1 and Rule 2 are used to obtain the conditions. In this case Rule 1 is tried first and if it contradicts previous conditions or if $\pi \cdot \pi' = 0$ is not preserved then Rule 2 is tried.

Given a partition π the method above finds a partition π', such that π and π' will satisfy the conditions of Theorem 3.2. Rules 1 and 2 simply place the states of M into blocks of π' such that condition c will be satisfied. This is done by forming a table of conditons for each state and arbitrary selecting one condition for each pair to find a partition π'. If conditions can be found for each state and $\pi \cdot \pi' = 0$ then Theorem 3.2 is satisfied.

The following example shows how π' can be found using the above rules. Examining the machine given in Table 3.2,

it can be shown that the partition $\pi = \{\overline{123};\overline{45}\}$ has SP. For this partition consider the block 123 and because $\#(\overline{123})=n(\pi)$ Rule 1 is used. Consider the pair (1,0) with $N(1,0)=4$. For this pair situation 2 of Rule 1 is used because one state, namely 3, exists such that $N(1,0)=N(3,0)$, see Table 3.2. To satisfy condition (c) of Theorem 3.1 for the pair (1,0), 1 and $N(1,0)$ is identified in π' or $3\equiv N(1,0)(\pi')$. These two identities are entered in the first row and first column of Table 3.3. The other conditions which block $\overline{123}$ places on π' are found in a similar fashion, and gives the entries of the first three rows of Table 3.3. To find the conditions placed on π' by the block $\overline{45}$, Rule 1 and Rule 2 are used, since $\#(\overline{45})<n(\pi)$ and $\overline{45}$ is mapped into $\overline{123}$ for inputs 0 and 1. Rule 1 is tried first and gives the remaining entries of Table 3.3. However, it is not possible to apply rule 1 to all pairs of the block $\overline{45}$ of π since this would give the identity partition. That is, from the pairs (1,0) and (4,0) the states 1,3 and 4 must be in the same block of π'. Also, from (2,0), (5,0) and (5,1) the states 2,3 and 5 must be in the same block of π'. This means that $\overline{134}$ and $\overline{235}$ are in the same block of π' (by the transitive law) implying that π' is the identity partition. This is not possible since $\pi \cdot \pi' = 0$ must be preserved. Therefore, rule 2 must be applied to block $\overline{45}$ of π. Rule 2 is tried on the pairs (4,0), (4,1), (5,0) and (5,1). It is impossible to apply Rule 2 to the pair (5,0) since this implies that states 2 and 5 would have to be in different blocks which

would contradict (2,0) by Rule 1. Rule 2 is applied to the pair (4,0) which states that state 3 must be placed in a different block than 4 and 5. The same conclusion follows from the application of Rule 2 to the pair (5,1). Using the condition obtained by Rule 1 from the block $\overline{123}$ and from the block $\overline{45}$ for the pairs (4,1) and (5,0), the following two partitions can be obtained; $\{\overline{14};\overline{25};\overline{3}\}$ and $\{\overline{34};\overline{25};\overline{1}\}$. From the pairs (4,0) and (5,1) and Rule 2 it may be concluded that the only partition that will satisfy Theorem 3.2(c) is $\pi' = \{\overline{14};\overline{25};\overline{3}\}$.

TABLE 3.2

State table of a synchronous machine

	0	0
1	4	1
2	5	2
3	4	3
4	3	1
5	2	3

TABLE 3.3

Identity table

	0	sit#	1	sit#
1	14 34	2	1	1
2	25	1	2	1
3	14 34	2	3	1
4	34	1	14	1
5	52	1	14	1

The state tables of the asynchronous and synchronous submachines can be found by the standard method as described in [10,17], using the partitions π and π'. Note that for some cases the state table produced by π' will have don't care conditions. It may be necessary to change a don't care condition to make it a stable state. The reason is that π' was found as a synchronous table and the tail machine must have a fundamental next state table.

Tables 3.4(a) and (c) can now be used to construct the decomposed circuit. The state table obtained from π is used to build the synchronous front machine and the state table obtained from π' is used to build the asynchronous tail machine.

TABLE 3.4

(a) State table of front machine
(b) State table of tail machine
(c) State table of tail machine
with out don't cares

	0	1
1	2	1
2	1	1

(a)

	10	11	20	21
A	A	A	C	A
B	B	B	B	C
C	A	C	-	-

(b)

	10	11	20	21
A	A	A	C	A
B	B	B	B	C
C	A	C	C	C

(c)

So far there has been no mention of the outputs; however, outputs can play a very important role in decomposition. Consider the output of the tail machine if it is realized by a circuit which is synchronous. Select a period of time starting at t^1 such that at t^n there is a transition from A to B. Let $t^1, t^2, \ldots, t^n, \ldots$ be the instants of time that the clock pulse changes from 1 to 0 (the trailing edge). Let the tail machine be stable from the time t^1 to t^{n+1} (in state A), and let the machine change states from A to B at t^{n+1}. Let M^1 be the synchronous front machine and M^2 the asynchronous tail machine. $O^2(K,A,I)$ will denote the output function of M^2 at any time t^n, where I is the input of M, and where K and A denote the present state of M^1 and M^2 respectively. If M^2 is realized by a circuit which is synchronous, the outputs of M^2 will remain constant from t^n to t^{n+1}. However, if M^2 is realized by a circuit which is asynchronous, the transition from A to B will take place at time t^n+d, where $d<t^{n+1}-t^n$ and denotes a short period of time. The output of the asynchronous circuit would be determined by state B from time t^n+d to t^{n+1}. Therefore, the external behavior of machine M is left unchanged if and only if for any state K of M^1 and any input I of M that produces a transition from state A to state B in M^2, $O^2(K,A,I)$ equals $O^2(K,B,I)$. This condition is formally stated in the following theorem.

THEOREM 3.3 [7] A sequential machine M has a non-trivial serial decomposition such that the tail machine can be realized by an asynchronous circuit without changing the output sequence of M if and only if two partitions π and π' of M exist, which satisfy Theorem 3.2., and in addition π satisfies the following condition:

> Any two states of M in a common block of π which have the same next state for some input I, also have the same output.

Proof: see [7]

To explain Theorem 3.3 consider the decomposition of a synchronous machine into a front machine which is synchronous and an asynchronous tail machine. For any input change that produces a transition in the synchronous front machine, the output of the tail machine must be constant during the transition.

To help in this type of serial decomposition, another type of synchronous sequential machine is defined [7]. This machine is defined so that the output function is a function of the next-state and input. That is, $Z(t) = O(S(t+1),I(t))$, where $S(t+1) = N(S(t),I(t))$ at any time t. This type of machine is called an anticipate machine and the structural model is given in Fig. 3.1.

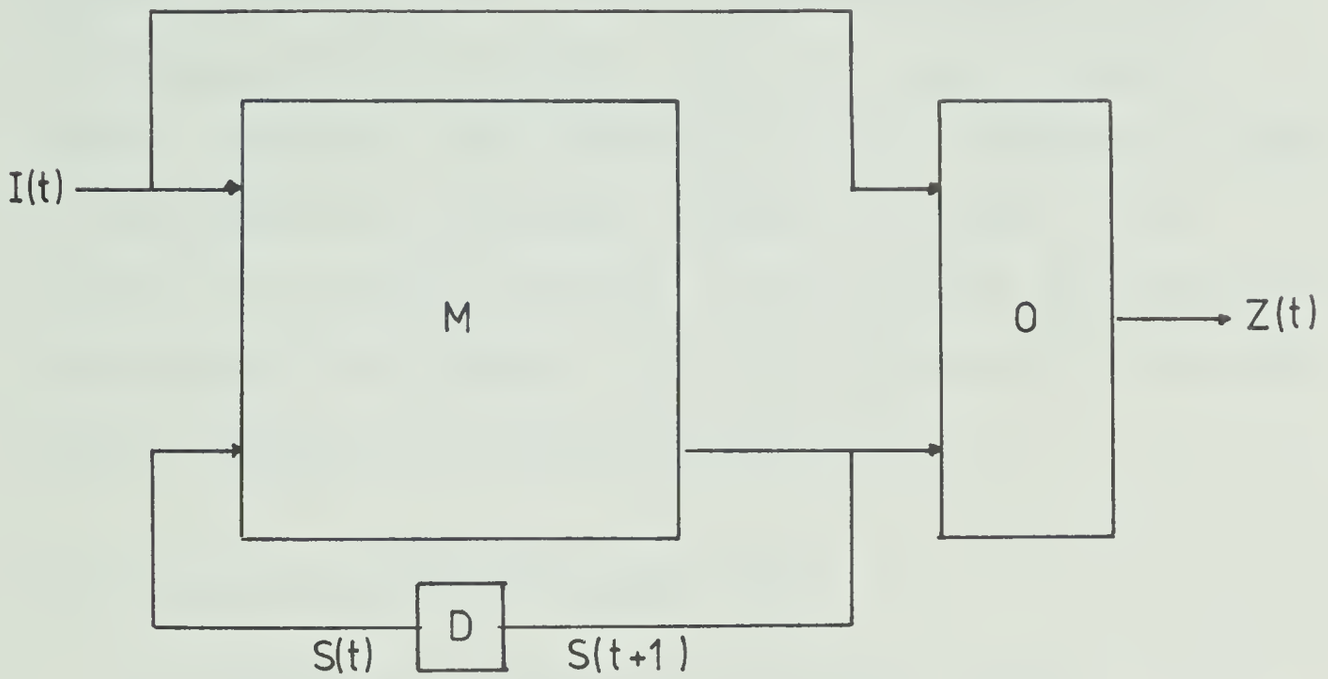


Figure 3.1

Anticipate Machine

An equivalent anticipate machine of M will exist if and only if those states of M which have the same next-state under the same input also produce the same outputs.

It becomes clear that if an anticipate synchronous machine can be realized as an asynchronous machine which preserves the state behavior, then the output sequence remains unchanged. The advantage of such a machine is that if two partitions satisfying Theorem 3.2 exist, it is possible to serially decompose the anticipate machine into a synchronous front machine and an asynchronous tail machine without changing the output sequence of M.

3.1.2 Asynchronous-Synchronous Case

To complete the study of synchronous serial decomposition, the case in which the front machine is asynchronous and the tail machine is synchronous is considered.

A partition π is said to be <u>fundamental</u> if and only if for any state a with any input I and $N(a,I) \in B$ (a block of π), then for all $b \in B$ the $N(b,I)$ are elements of B. The above definition enables us to give conditions on the partitions and outputs, so that a serial decomposition exists with an asynchronous machine controlling a synchronous machine. The conditions are stated in the following theorem.

THEOREM 3.4 [8] For a sequential machine M a serial decomposition exists, such that the front machine can be realized by a synchronous circuit, if two partitions π,π' exist which satisfy the following conditions:

a. $\pi \cdot \pi' = 0$

b. π is a fundamental SP partition

c. Any two states of M in a common block of π' which have their next-states in the same block of π for some input I, must have the same next state for this I.

d. Any two states of M in a common block of π' which have their next-states in the same block of π for some I, must have the same output state for this I.

Proof: see [8]

To explain Theorem 3.4 consider the decomposition of a machine M into two synchronous machines. Let the front machine consist of combinational logic to determine the y-variables and a clocked delay for each y-variable. The delay element merely delays the output from the combinational logic one clock pulse, as seen in Fig. 3.2. Consider the realization of the front machine by an asynchronous circuit. As discussed in section 3.1.1, the state table of M^1 must be fundamental in order for the two serially connected machines to have the same state behavior as M. This is established by condition (b) of the above

theorem . Considering the outputs, let t^n denote a time when the inputs change, thus M^1 changes states at time t^n+d, where d is the delay time through the combinational circuit. If the clocked delays were present M^1 would change state at time t^{n+1}. In order to realize the front machine by an asynchronous circuit without changing the behavior of M, it must be ensured that the state behavior of the tail machine and the output function are independent of any change of M^1 from time t^n+d to t^{n+1}. Part (c) and (d) of the theorem state the conditions in which this holds.

Essentially, for this type of decomposition, all that is needed is to find two partitions π and π' satisfying Theorem 3.4. The best method of finding the two partitions is to consider all pairs of partitions of M, and checking each pair to see if the theorem is satisfied. If a pair of partitions exists then the asynchronous front machine can be realized directly from the partition π.

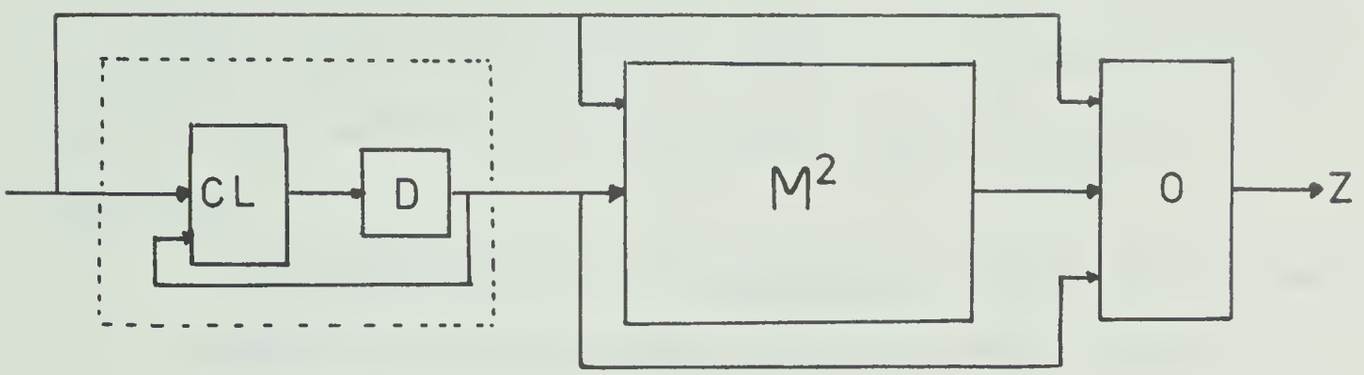


Figure 3.2

Serial Decomposition of Synchronous Machine

3.2 Asynchronous Decomposition

The theory of synchronous decomposition has been extended by Tan et al. [30] to that of asynchronous decomposition. The theories are further extended so that one of the submachines of the decomposition can be realized by a circuit which is synchronous.

Before proceeding with the method for decomposition of asynchronous machines, the following definitions are essential.

Definition 3.1 [30] A transition time is the maximum amount of time necessary for any y-variable to change values.

Definition 3.2 [30] A state assignment for which a single transition time is always sufficient for any transition is called a single transition time (STT) assignment.

Definition 3.3 [30] Let U and V be disjoint subsets of the states of the machine. The unordered pair (U,V) is referred to as a partial state dichotomy (or simply dichotomy) of the machine.

Definition 3.4 [17] Two states s_i and s_j of machine M are compatible if and only if for all input sequences they produce the same output.

Definition 3.5 [17] A set of states is called compatible if all its members are compatible.

Definition 3.6 [17] A compatible is called maximal

if it is not covered by any other compatible.

A state variable y_i is said to cover a dichotomy if $y_i=0$ for every state of U and $y_i=1$ for every state of V, or vice versa. Given a pair of transitions under the same input i->j and k->m, (ij,km) is the dichotomy associated with the transition. If i=j then (i,km) forms a degenerate dichotomy, and if both transitions are equal the associated dichotomy reduces to (i,k).

Partition theory is used for the decomposition of asynchronous sequential machines, as was the case for synchronous decomposition. For the asynchronous case, as well as synchronous, a necessary condition for serial decomposition is the existence of a non-trivial SP partition π. This partition defines the front machine M^1 and when connected in series with the tail machine M^2, the two machines will realize M.

The major difference between serial decomposition of asynchronous and synchronous machines is the method of finding the tail machine. In synchronous decomposition the submachines are assumed to operate simultaneously, but this assumption is not valid for the asynchronous case. A solution can be obtained by ensuring that the submachines are not dependent on the order in which the states change. To make the submachines independent of the order of change, the submachines are made to operate in a single transition time by using STT assignments.

The procedure for asynchronous decomposition is now given. If there exists an SP partition π, then a flow table for M^1 is constructed [17]. A race free SST assignment for M^1 may be found using Tracey's algorithm [30] or any other suitable method [33]. The state assignment for M^1 is used as a partial state assignment for M. Additional state variables are assigned to cover the dichotomies of M not already covered by the state variables of M^1. Let this set be Q. The state variables of Q so obtained may be considered as the state variables of M^2.

The best method of finding the flow table of M^2 is to list all the dichotomies of M not covered by the state variables of M^1. A minimal set of maximal compatibles covering these dichotomies is found. Let this set be denoted by R, with elements r_i, i=1,... n, then the partition π' defined as the product of r_i, for all i. The state table for the tail machine M^2 can be found by forming a table whose rows correspond to the blocks of π' and whose inputs are all combinations of inputs and state variables of M^1. The entries are filled in by noting the required transitions of M and observing that the M^2 entries must be filled in such that M^2 state changes can occur either before or after M^1 state changes.

The procedure is illustrated using machine M given in Table 3.5(a). The two partitions $\pi_1=\{\overline{123};\overline{45};\overline{67};\overline{89}\}$ and $\pi_2=\{\overline{12367};\overline{4589}\}$ have SP on the set of states of machine M.

The partition π cannot be used to obtain a nontrivial decomposition using STT assignments because it does not satisfy any of the dichotomies of machine M. The dichotomies that must be covered are (1,47) (15,24), (1,56),..., (28,39). Partition π_1 is used to define the front machine M^1 of the decomposition, since π_2 does satisfy some of the dichotomies of M. Table 3.5(b) gives the resulting machine M^1. An STT assignment is found for M^1, using Tracey's method [31], and is used as a partial assignment for M, as shown in Table 3.5(b). π' is found by considering the dichotomies that are not covered by π. These are; (1,28), (1,39), (28,39), (47,56), (15,24), (24,35) and (68,79). One minimal set of maximum compatibilities covering these dichotomies consists of (147,235689) and (13579,2468), and π' is defined as their product, i.e., $\pi' = \{\overline{147},\overline{23689}\} \bullet \{\overline{13579},\overline{2468}\} = \{\overline{17};\overline{4};\overline{359};\overline{268}\}$. The state table for M^1 is defined using π_1; the result is given in Table 3.5(b). To find the state table for M^2 the partition π' is used. Consider as an example the determination of the entry in the state table of M^2 for input a0 and state e. The front machine M^1 is in state a with input 0, thus machine M must be in either states 1,2 or 3. If machine M^2 is in state e, M must be in states 1 or 7. Machine M must be in state 1, since M^1 is in state a and M^2 is in state e. The transition from state 1, input 0 of M, is to state 1, therefore, the transition of M^2

TABLE 3.5

(a) State table for machine M
(b) State table of front machine

	0	1	$y_1y_2y_3y_4$
1	1	5	0 0 0 1
2	2	4	0 0 1 0
3	3	5	0 0 1 1
4	7	4	0 1 0 0
5	6	5	0 1 1 1
6	6	8	1 1 1 0
7	7	9	1 1 0 1
8	2	8	1 0 1 0
9	3	9	1 0 1 1

(a)

	0	1	y_1y_2
(123) a	a	b	0 0
(45) b	c	b	0 1
(67) c	c	d	1 1
(89) d	a	d	1 0

(b)

TABLE 3.6

State table of tail machine M^2

	a0	b0	c0	d0	a1	b1	c1	d1
(17) e	e	-	e	-	g	-	g	-
(268) f	f	f	f	f	h	-	f	f
(359) g	f	-	g	g	g	g	-	g
(4) h	-	e	-	-	-	-	h	-

must be to the block containing state 1, which is block e, thus N(e,a0)=e. Likewise, considering the b0 input to M^2, machine M^1 is in block b, thus M is in either state 4 or 5. For M^2 to be in state e with input b0, machine M must be in either state 1 or 7. Since this is not possible the entry in row e column b0 is made a don't care. Table 3.6 is the state table of M^2 with all the entries filled in as explained above. Table 3.6 would, as it stands, operate as if both submachines change state concurrently, since the decomposition is basically the same as for the synchronous case. In as much as the two machines are not dependent on the order of change, some of the don't cares must be changed so that the appropriate transitions can be made. The don't cares are changed by noting the required transitions of M and observing that the M^2 entries are filled in so that an M^2 state change can occur before or after an M^1 state change.

As an example, consider the transition 1->5, with an input of 1. The front machine M^1 will have the transition a->b. If M^2 detects the change in y_2 and y_3 before it receives the input change, then state e under input a0 goes to e under input b0; therefore, the don't care entry under input b0 of Table 3.6 is changed to an e. After this change is made, M^2 can now make the required transition, i.e., e(a,0)->e(b,0)->e(b,1)->g(b,1). All transitions must be checked where M^2 might receive the y_2, y_3 changes before M^2 detects the input change. Similarly, all transitons where

M^2 might receive the change in the inputs before it receives the y_2 and y_3 changes must be checked and the appropriate don't cares changed. The resulting flow table for the tail machine M^2, is given in Table 3.7.

TABLE 3.7

Final state table of M^2

y_3y_4	a0	b0	c0	d0	a1	b1	c1	d1
0 0 e	e	e	e	-	g	g	g	g
1 1 f	f	f	f	f	h	h	f	f
1 0 g	f	f	g	g	g	g	g	g
0 1 h	-	e	e	-	h	h	-	-

The above method produces the state table of the tail machine, such that it is not dependent on the order in which the two submachines change states. However, by restricting the order of state change, it is often possible to obtain a simpler realization. A restricted order of state change for the submachines will also be useful in realizing one of the components by a synchronous circuit.

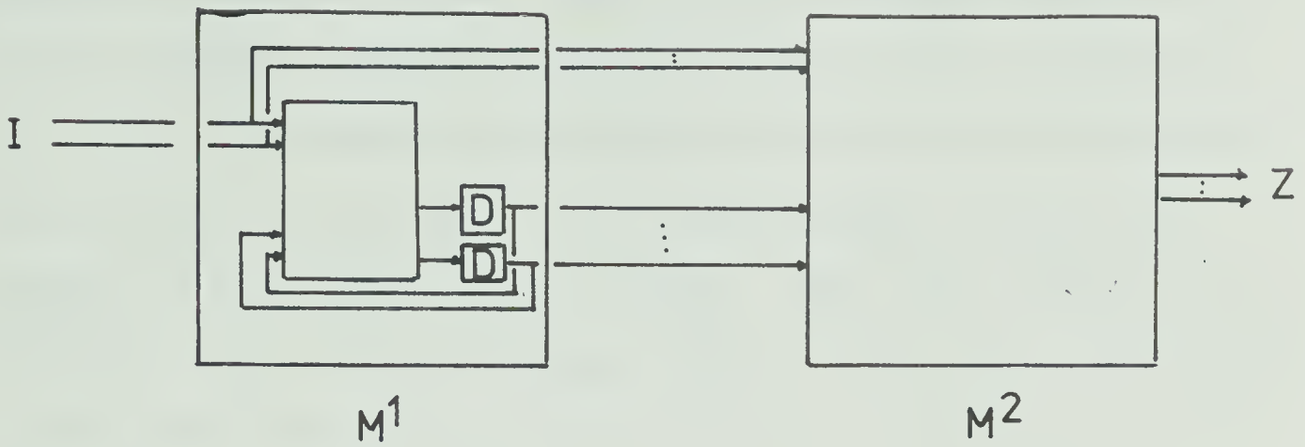


Figure 3.3

Serial Decomposition in which M^2 changes first

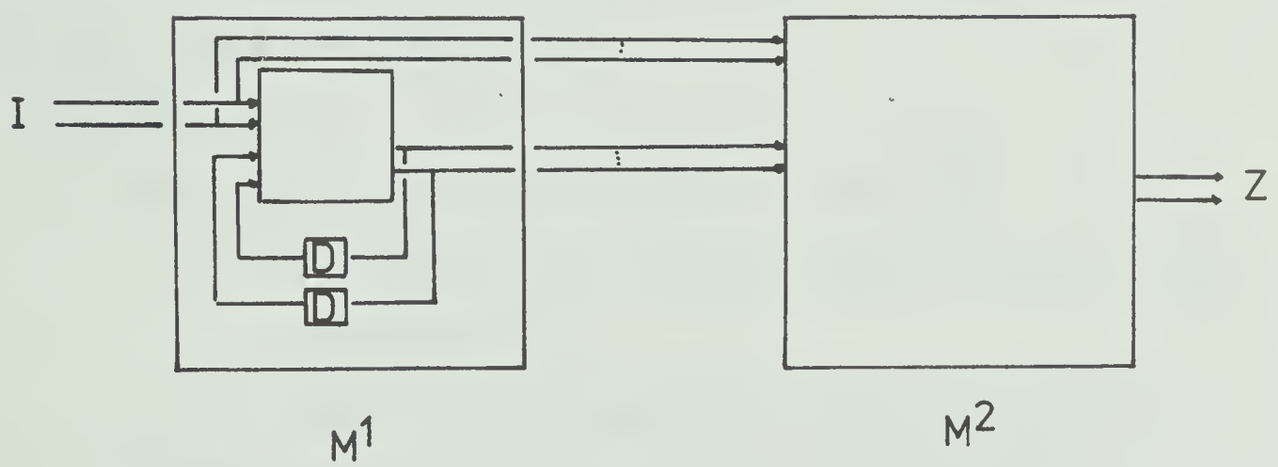


Figure 3.4

Serial Decomposition in which M^1 changes first

There are two methods to restrict the order of change of the submachines. One is to decompose M so that the tail machine M^2 changes first. That is, the present state of M^1 and the inputs determine the next state of M^2, as seen in Fig. 3.3. The delays of M^1 should be such that M^2 reaches a stable state in response to the input change, before the outputs of the delays in M^1 change. The other method is to decompose M so that M^1 changes first. M^2 is then determined by the inputs and the next state of M^1, as seen in Fig. 3.4.

The front machine M^1 is found using the partition π as it was done in section 3.1.1. The tail machine M^2 is found by the procedure given below. This procedure uses the notation B_i and B_j to denote blocks of π such that B_i contains state i and B_j contains state j. B_i and B_j therefore denote states of M^1.

PROCEDURE 3.1

1. Construct a new state table M" such that
   a. The inputs are combinations of inputs I and states of M^1.
   b. The rows correspond to every state of M.
2. Entries are made in M" for all transitions i->j in M as follows;
   a. If M^2 is required to change before M^1, then M" is constructed such that $N(i,(B_i,I^K))=j$, $N(j,(B_i,I^K))=j$, and $N(j,(B_j,I^K))=j$.
   b. If M^1 is required to change first then M" is constructed such that $N(i,(B_j,I^K))=j$ and

$$N(j,(B_j,I^K))=j.$$

3. Reduce M" by any state reduction technique [1,3,10,17], making sure that no pair of states contained in the same block of π are merged. The reduced flow table defines the tail machine M^2.

The entries in M" are made according to the model of serial decompostion chosen, so the the series connection of M^1 and M" realizes M. A state table M" can always be found using this procedure; however, if M" cannot be reduced then the trivial serial decomposition exists, i.e., $M^2=M$.

Procedure 3.1 is illustrated using machine M given in Table 3.5(a). The flow table for M^1 is given in Table 3.5(b). If M^2 is required to change before M^1, then the equation given in Procedure 3.1, step 2a is used to find the state table M". As an example, consider machine M with the transition 1->5 and input equal to one. In this case the following entries are made: N(1,(a,1))=5, N(5,(a,1))=5 and N(5,(b,1))=5 (these entries are marked with a star in Table 3.8(a)). The M" state table is obtained by considering all transitions of M and using the equations of Procedure 3.1, step 2a. M" is then reduced, resulting in Table 3.8(b). An STT assignment is found for M^2, noting that the dichotomies (e,fg), (f,eg) and (g,ef) are the only ones that have to be covered. The three variable assignment shown in Table 3.8(b) is obtained by letting y_1 cover (g,ef), y_2 cover (f,eg) and y_3 cover (e,gf). A valid non-STT assignment

could also be used since the order of change of the two submachines is restricted.

TABLE 3.8

(a) State table M"
(b) Reduced state table M"

	a0	b0	c0	d0	a1	b1	c1	d1
1	1	-	-	-	5*	-	-	-
2	2	-	-	2	4	-	-	-
3	3	-	-	3	5	-	-	-
4	-	7	-	-	4	4	-	-
5	-	6	-	-	5*	5*	-	-
6	-	6	6	-	-	-	8	-
7	-	7	7	-	-	-	9	-
8	-	-	-	2	-	-	8	8
9	-	-	-	3	-	-	9	9

(a)

	a0	b0	c0	d0	a1	b1	c1	d1	Y_3 Y_4 Y_5
(156) e	e	e	e	-	e	e	f	-	0 1 1
(248) f	f	g	-	f	f	f	f	f	1 0 1
(397) g	g	g	g	g	e	-	g	g	1 1 0

(b)

TABLE 3.9

(a) State table of M"
(b) Reduced state table M" giving the tail machine M^2

	a0	b0	c0	d0	a1	b1	c1	d1
1	1	-	-	-	-	5*	-	-
2	2	-	-	-	-	4	-	-
3	3	-	-	-	-	5	-	-
4	-	-	7	-	-	4	-	-
5	-	-	6	-	-	5*	-	-
6	-	-	6	-	-	-	-	8
7	-	-	7	-	-	-	-	9
8	2	-	-	-	-	-	-	8
9	3	-	-	-	-	-	-	9

(a)

	a	b	c	d	y_3	y_4	y_5
(156) e	e	e	e	f	1	1	0
(248) f	f	f	g	f	0	0	0
(379) g	g	e	g	g	1	0	1

(b)

If M^1 is required to change before M^2 then the state table of M" is found by using the equations given in Procedure 3.1, step 2b. Considering the same transition as above, the entries N(1,(b,1))=5 and N(5,(b,1))=5 are made in the M" state table. The M" flow table is completed in this manner for all transitions in M, giving the state Table 3.9(a). M" is reduced to give the resulting tail machine M^2 as specified by Table 3.9(b). An STT assignment or any valid state assignment free of critical races can be used as the state assignment for M^2.

3.2.1 Synchronous-Asynchronous

This section looks at the problem of decomposing an asynchronous machine into a synchronous front machine and an asynchronous tail machine. Recall Tan's [30] decomposition procedure and the two solutions that exist. The first solution was to restrict the operation of the submachines so that the tail machine makes a state change before the front machine. It is a trivial matter if this procedure is used to replace the asynchronous front machine with a synchronous circuit. The state table of the front machine is used directly for the synchronous circuit. This is possible because the resulting flow tables operate in normal mode. It should be noted that the two submachines must also satisfy the conditions stated in section 2.2.

The outputs of the combined machine can be considered

in the same manner as for the synchronous case in section 3.1.1. In particular Theorem 3.3 specifies the properties which the state table must possess so that the external behavior of the machine M will be left unchanged.

3.2.2 Asynchronous-Synchronous

Tan's decomposition procedure can be used to decompose an asynchronous machine into an asynchronous front machine and a synchronous tail machine. The second solution for asynchronous decomposition, given by Tan, restricts the state change of the tail machine so that the front machine changes first. In this type of decomposition the state table of the tail machine can be directly realized with a synchronous circuit.

In order to keep the outputs of the decomposition consistent with the outputs of the original machine the conditions given by Theorem 3.4 must be satisfied. For both cases, asynchronous-synchronous and synchronous-asynchronous, the conditions and restrictions that were given in Chapter 2 must also be observed.

3.3 Parallel Decomposition

This section studies the decomposition of one machine into two machines that operate in parallel. The discussion will be restricted to the study of parallel state behavior decompositions. The condition for the existence of a

parallel decomposition is given in the following theorem:

> THEOREM 3.5 [13] A sequential machine M has a nontrivial parallel decomposition of its state behavior if and only if there exist two nontrivial SP partitions π and π' on M such that $\pi \cdot \pi' = 0$.
>
> Proof: see [13].

Synchronous parallel decomposition is possible if two SP partitions exist, such that the product of these partitions equals the zero partition. The partition can be found as described in [18]. Each partition is then used to realize a synchronous component of the decomposition.

The conditions necessary for the existence of a parallel decomposition of a synchronous machine into synchronous and asynchronous components are obtained by combining Theorems 3.2 and 3.5. In other words, Theorem 3.2 with the further condition that π' have SP, states the conditions for parallel decomposition of this type. To decompose a synchronous machine into synchronous and asynchronous submachines, all pairs of SP partitions (π^1, π^2) must be found such that $\pi^1 \cdot \pi^2 = 0$. These pairs are then examined to verify if either π^1 or π^2 satisfy condition (c) in Theorem 3.2. If such a pair of partitions exists then the partition that represents π' in Theorem 3.2 can be realized by an asynchronous circuit.

The condition for the existence of an asynchronous parallel decomposition is given in Theorem 3.5. That is,

the asynchronous case is exactly the same as the synchronous case. In order to have a parallel decomposition of an asynchronous machine into asynchronous or synchronous submachines it is assumed that the machine to be decomposed operates in normal mode. Considering only this type of machine the submachines of the decomposition will also operate in normal mode. This means that both submachines reach a stable state after each input change, therefore either one of the asynchronous state tables can be used to realize one of the submachines with a synchronous circuit.

The above discussion only considers the state behavior of the machine; however, if the outputs are to be considered the following is needed. The condition stated in Theorem 3.3 must be imposed if the outputs are to be time dependent. With this further condition M can be decomposed to retain the same output behavior of M. Again it should be noted that the two submachines must satisfy the conditions stated in section 2.1.

3.4 Summary

In this chapter, decomposition of both synchronous and asynchronous machines was discussed. Serial as well as parallel decomposition was studied. Of particular interest was the decomposition of a machine of one type into two submachines, one of which is synchronous and the other asynchronous. Generally decomposition of machines will

lower the cost of the combinational part of the machine [17]. Apart from the economical realization of sequential machines, the decomposition studied here is useful for a better understanding of the role that delay elements play in synchronous circuits. This understanding is useful in realizing a hybrid machine from a synchronous circuit. The next chapter will concentrate on this problem.

CHAPTER 4

HYBRID MACHINES

The two types of sequential machines are now brought together to form a new type, the hybrid machine. In Chapter 1 it was stated that a hybrid machine has clocked memory elements in some but not all of the feedback paths. In this chapter two methods are presented for forming a hybrid machine. One is to remove memory elements from synchronous networks, the other is to insert clocked memory elements in asynchronous networks.

4.1 Removal of Delay Elements From Synchronous Circuits

Two approaches will be presented for the removal of clocked delay elements from synchronous machines. The first begins with a circuit and finds the conditions when delay elements can be removed without changing the behavior. The other approach begins with the state table of a synchronous machine and finds a state assignment that will have the maximum number of removable delays. This will be accomplished by expressing the conditions needed for the

state assignment in terms of partitions.

Before pursuing the problem of removing delay elements, recollect from Chapter 1 the model of a sequential machine and that in a synchronous machine all delay elements are clocked. In other words, the next-state is simply delayed by one clock period. Throughout this chapter a clocked delay element will be referred to simply as a delay element, and it is implied that all delay elements are synchronized. T will be used to denote the period of one clock pulse and Y_i'-y_i will be used to denote the i-th feedback path.

Consider now, what would happen if one of the delay elements is removed from a feedback path. For a synchronous machine any internal variable is subject to, at most, a single transition in the interval T. Removal of one of the delays, say D_i, results in three possiblities:

1. No futher change in the total state occurs. This happens when the present state variable y_i equals the next state variable Y_i'. In this case the behavior of the synchronous machine will not be affected.
2. The variable y_i will oscillate between 0 and 1. In this case the behavior of the machine is dependent on the state the y_i variable is in when the next clock pulse occurs.
3. The state of y_i changes. In other words the next state variable Y_i' will become the present state

variable y_i in time t, for $0<t<T$, and stablilizes. In order that the behavior of the circuit be deterministic, results 1 or 3 must occur for all states of the circuit.

In all the cases the only delay in the Y_i-y_i feedback path is caused by stray delays in the circuit. It is assumed that all stray delays will be less than T. This means that the variable y_i will change before all other y-variables and that the input to the combinational network will be subject to change before the next clock pulse. Thus, the combinational network will determine a new total state for this new input, which implies that the variable y_i can make more than one transition in the interval T. This new total state will be referred to as the hybrid transition state. In order to preserve the behavior of the machine, it is essential that the hybrid transition state produces no further transition in the outputs of the combinational circuit. If this happens the value of y_i is said to be stablilized in the feedback path until the next clock pulse.

The non-delayed internal variables will be referred to as the asynchronous variables, whereas, the synchronous variables are those that contain clocked delay elements in their feedback paths. Consider what would happen if there was more than one asynchronous variable. In order that the external behavior be unaffected by the change of the asynchronous variables the outputs must not be dependent on the changes of these variables. This result is formally

summarized by the following definitions and theorem.

The hybrid transition state H[A] associated with the total state A of a synchronous circuit is the ordered set of next-state, present state and input variables $Y_1',\ldots,Y_h'$, $y_{h+1},\ldots,y_s,x_1,\ldots,x_n$. The variables $Y_1',\ldots,Y_h'$ of H[A] are the h next-state variables of A, i.e., the asynchronous variables. The variables $y_{h+1},\ldots,y_s$ are the s-h present state variables of A, i.e., the synchronous variables. In contrast, the hybrid transition state associated with the total state A of an asynchronous circuit is the ordered set of present state, next-state and input variables, $y_1,\ldots,y_h,Y_{h+1}',\ldots,Y_s',x_1,\ldots,x_n$. The $y_1,\ldots,y_h$ variables of H[A] are the h present state variables of A and the $Y_{h+1}',\ldots,Y_s'$ variables are the s-h next-state variables of A.

> Theorem 4.1 [9] The clocked delay elements D_h, $1<h<s$, in the feedback paths $Y_1'-y_1,\ldots,Y_h'-y_h$ of a synchronous machine M can be removed if and only if, for any total state A which produces the output O(A), the associated hybrid transition state H[A] also produces the output O(A).
>
> Proof: see [9].

To explain the theorem consider the synchronous machine defined by Table 4.1. Let the delay elements in the feedback paths $Y_1'-y_1$ and $Y_3'-y_3$ be the ones that are to be eliminated, i.e., D_1 and D_3 are to be removed. From the

theorem the hybrid transition state associated with A is $H[A] = Y'_1, y_2, Y'_3, x$. Consider the total state A=0,0,0,0 giving the total output state O(A)=1,1,1,0,1 and the associated hybrid transition state H[A]=1,0,1,0. From the transition table O(H[A])=1,1,1,0,1 which clearly equals O(A). Therefore, for this transition the delay elements D_1 and D_3 are not necessary; however, all total states of the table must be checked. If all states satisfy the theorem then the delays D_1 and D_3 can be removed without changing the behavior of the machine. It can be shown that all total states of Table 4.1 satisfy Theorem 4.1. Thus, a hybrid machine exists for this transition table.

One method for removing delay elements in a machine is to check that all total states satisfy Theorem 4.1. However, if the number of states in the transition table is not equal to 2^p, then the next-state of some of the associated hybrid transition states might not be specified. In order for the machine to function correctly these states will have to be specified. The following algorithm gives a method of converting a synchronous machine to a hybrid one and takes into account the filling in of the unused states. The algorithm is dependent on the state assignment; therefore, the transition and output tables are used.

TABLE 4.1

Transition table to illustrate Theorem 4.1

	0	1	0	1
0 0 0	111	010	01	11
0 1 0	000	010	11	01
1 1 0	100	101	00	01
1 0 0	110	110	01	00
0 0 1	011	110	11	00
0 1 1	001	010	10	01
1 1 1	001	101	10	01
1 0 1	111	111	01	10
	$Y'_1\ Y'_2\ Y'_3$		$z_1 z_2$	

Algorithm 4.1

Let: L denote the set of integers 1,...,s,

K denote the collection of $\binom{s}{h}$ distinct combinations of L,

k_i denote the elements of the set K, for $1<i<\binom{s}{h}$.

1. Set h=s.
2. If machine M has a fundamental state table then go to step 11.
3. Set h=h-1, find the set K for this h.
4. Set i=1.
5. Calculate A, the associated hybrid transition state H[A] and O(A) as specified in Theorem 4.1 for

feedback paths Y'_j-y_j, $j=k_i$, and any transition.

6. If next-state of H[A] is not specified then specify it with the same next-state and output of A, otherwise compare the output produced by H[A] to O(A), if they are not equal, go to step 8.
7. If A, H[A] and O(A) are calculated for all total states, go to step 10, otherwise calculate H[A] and O(A) for a new total state A and go to step 6.
8. Set i=i+1
9. If $i>\binom{s}{h}$ go to step 12 otherwise go to step 5.
10. The h delay elements D_j, $j=k_i$, can be removed, and the process stops.
11. All s delay elements can be removed, and the process stops
12. If h>1 then go to step 3.
13. No delay elements can be removed, and the process stops.

For a given state assignment the algorithm will always find a hybrid circuit if it exists. The reason is that the algorithm is a finite process that checks all the sets of delays to determine if they can be removed. For each set all the states are examined and if each state satisfies Theorem 4.1 then the algorithm stops. Otherwise, a new set is checked and this process continues until a hybrid circuit is found or no delays can be removed.

The following example illustrates, in detail, the

application of the algorithm to the synchronous machine given in Table 4.2(a).

Example 4.1

1. Set h=3

2. Machine M does not have a fundamental state table, therefore all the delay elements cannot be removed.

3. h=3-1=2 and K = (12,23,13).

4. i=1.

5-6-7. Is Theorem 4.1 satisfied if D_1 and D_2 are removed? Consider the total state A=0,0,0,0 giving the output O(A)=1,1,1,0,0, and the associated hybrid transition state H[A]=1,1,0,0. O(H[A]) must be the same as O(A) if D_1 and D_2 are to be removed. From Table 4.2 O(H[A])=0,1,0,0,0 which differs from O(A); therefore D_1 and D_2 cannot be removed.

8. i=1+1=2.

9. i>3, therefore go to 5.

5-6-7. Is Theorem 4.1 satisfied if D_2 and D_3 are removed? Checking all transitions, it can be seen that Theorem 4.1 holds for this case. For example, consider the total state A=0,0,0,0 if D_2 and D_3 are removed then the associated hybrid transition state H[A]=0,1,1,0. The total output state of A is O(A)=1,1,1,0,0 and the next-state of H[A] is not specified, therefore it must be specified with N(A) and the output of A. Thus, N(H[A])=1,1,1 and the output is 0,0, this is shown in Table 4.2(b). Consider the total state A=0,0,1,0 with

the associated hybrid transitions state H[A]=0,0,1,0, this results in O(A) being equal to O(H[A]) because A=H[A]. It can be seen that the same holds for all total states of the circuit.

10. The process stops with the result that D_2 and D_3 can be removed.

TABLE 4.2

(a) Transition table to illustrate Algorithm 4.1
(b) Transition table of hybrid circuit
(y_2 and y_3 are delay free)

	0	1	0	1
0 0 0	111	100	00	01
0 0 1	101	110	10	10
0 1 0	110	110	01	10
0 1 1	---	---	--	--
1 0 0	---	---	--	--
1 0 1	001	000	11	11
1 1 0	010	110	00	01
1 1 1	001	000	11	11

$Y'_1\ Y'_2\ Y'_3$ $z_1 z_2$

(a)

	0	1	0	1
0 0 0	111	100	00	01
0 0 1	101	110	10	10
0 1 0	110	110	01	10
0 1 1	111	---	00	--
1 0 0	---	000	--	11
1 0 1	001	000	11	11
1 1 0	010	110	00	01
1 1 1	001	000	11	11

$Y'_1\ Y'_2\ Y'_3$ $z_1 z_2$

(b)

It should be noted that the algorithm is dependent on the the state assignment. A change in the assignment could have the result that different delay elements are removable. The possibility that no delays are removed can also result. As an example, consider Table 4.1 and change the state assignment to that given in Table 4.3. If the algorithm was applied to this table the result is that no delays can be removed. This differs from the result that two delays can be removed from Table 4.1.

TABLE 4.3

Transition table equivalent to Table 4.1

	0	1	0	1
0 0 0	110	001	01	11
0 0 1	000	001	11	01
0 1 0	011	011	00	01
0 1 1	010	010	01	00
1 0 0	101	010	11	00
1 0 1	100	001	10	01
1 1 0	100	111	10	01
1 1 1	110	110	01	10
	$Y_1' Y_2' Y_3'$		$z_1 z_2$	

Algorithm 4.1 shows how it is possible to minimize the number of delay elements for a given state assignment. However, different state assignments might lead to a different number of removable delays and one specific state assignment might not yield the maximum number. The following theorem gives the conditions needed that will give the largest number of removable delay elements.

Theorem 4.2 [9] For a synchronous machine M with p internal states, an assignment having $s = \lceil \log_2 p \rceil$ variables giving a realization with s-h delay elements exists, if and only if there exists a pair of partitions (π', π) on the set of internal states of M satisfying the following conditions:

1. $\pi \cdot \pi' = 0$, $\lceil \log_2 n(\pi') \rceil = h$, and $\lceil \log_2 n(\pi) \rceil = s-h$.
   Where $\pi' = \{A^1, A^2, \ldots, A^m\}$,
   and $\pi = \{C^1, C^2, \ldots, C^n\}$.
2. Let $s_i \in A^i$, $s_i \in C^i$ and $N(s_i, I^k) \in A^r$.
   If $A^r \neq A^i$ then:
   a. If $A^r \cap C^i \neq \emptyset$, the state s_j, where $s_j = A^r \cap C^i$, satisfies $N(s_j, I^k) = N(s_i, I^k)$ and $O(s_j, I^k) = O(s_i, I^k)$
   b. If $A^r \cap C^i = \emptyset$, and for the same input there exists some other state s_p such that $s_p \in C^i$, and $N(s_p, I^k) \in A^r$ then:
   $N(s_p, I^k) = N(s_i, Ik)$ and
   $O(s_p, I^k) = O(s_p, I^k)$.

Proof: see [9].

To explain the theorem, consider Table 4.4 and the two partitions $\pi' = \{\overline{1}; \overline{23}; \overline{45}; \overline{67}\}$ and $\pi = \{\overline{347}; \overline{1256}\}$. Thus, $\pi \cdot \pi' = 0$, $\lceil \log_2 n(\pi') \rceil = 2$ and $\lceil \log_2 n(\pi) \rceil = 1 = (s-h)$ which for this example satisfies condition 1. To illustrate condition 2, consider s_i to be state 3 existing in block $\overline{23}$ of π' and in block $\overline{347}$ of π, with N(3,0) existing in block $\overline{45}$ of π'. Clearly $\overline{23} \neq \overline{45}$ and condition 2a ($\overline{45} \cap \overline{347} = 4 \neq \emptyset$) implies that state 4 must satisfy N(4,0)=N(3,0) and O(4,0)=O(3,0). Obviously from Table 4.4 these conditions hold. Consider now state 4, input 1 with N(4,1)=1 existing in block $\overline{1}$ of π'. Thus, $\overline{1} \cap \overline{347} = \emptyset$ and by condition 2(b) if there exists a state s_p in block $\overline{347}$ of π with $N(s_p,1) \varepsilon 1$ then $N(s_p,1)$ must equal N(4,1) also, $O(s_p,1)$ must equal O(4,1). The states that exist in $\overline{347}$ with next-state equal to 1 are 3,4 and 7, therefore, it is necessary to check that the condition 2(b) is satisfied. In other words, N(3,1)=1=N(4,1) and O(3,1)=01=O(4,1) also, N(7,1)=1=N(4,1) and O(7,1)=01=O(4,1). As can be seen condition 2b is satisfied for the total state (4,1); however, it is necessary to exhaustively check all states in order to find out if π' and π satisfy Theorem 4.2.

Essentially all this theorem states is that if two partitions satisfying conditions 1 and 2 exist, then a state asssignment derived from the two partitions will realize a hybrid machine with the maximum number of delay elements removed. A method of finding a pair of partitions (π', π) satisfying this theroem is now presented.

TABLE 4.4

Synchronous next-state table used to illustrate Theorem 4.2

	0	1	0	1
			z_1z_2	
1	2	2	11	10
2	3	7	00	00
3	5	1	10	01
4	5	1	10	01
5	1	4	01	11
6	2	7	11	00
7	6	1	01	01

Consider a sequential machine M with p internal states, where $p=2^s$. A method of obtaining the pair of partitions (π',π) consists of the following two steps:

Step 1: Examine each total state (s_i,I^K) of the flow table and derive a table, called a condition table, which has as its rows the states of the flow table and two columns (one for π' and one for π). A condition table is formed for each input I^K. The condition table lists the possible alternatives satisfying condition 2 of Theorem 4.2. Entries are made by considering the following situations:

1. If no present state s_j exists such that $N(s_j,I^K)=N(s_i,I^K)$ and $O(s_j,I^K)=O(s_i,I^K)$ then identify s_i and $N(s_i,I^K)$ in π', i.e., $(s_i,(N(s_i,I^K))$

is entered in a row of s_i in the condition table under column π' of table I^K.

2. If m states s_j, for j=1,...,m, exist such that $N(s_j,I^K)=N(s_i,I^K)$ and $O(s_j,I^K)=O(s_i,I^K)$ then the following entries are made in the condition table;

    1. s_i and $N(s_i,I^K)$ are identified under column π' of table I^K.

    2. $N(s_i,I^K)$ and s_j are identified under column π' and s_i and s_j are identified under column π of table I^K, for all j=1,...,m

It must be noted that if $N(s_i,I^K)=s_i$ as in case 1 and if $N(s_i,I^K)=s_j$ as in case 2.2, there is no effective condition of identification on π'. Also, if in case 2.2 if $N(s_i,I^K)=N(s_j,I^K)=s_i$ no conditon of identification is made since $N(s_i,I^K)$ and s_j are identified in π' and s_i and s_j (which implies $N(s_i,I^K)$ and s_j are identified in π). This violates the conditon that $\pi \cdot \pi' = 0$.

In order to obtain the condition tables, all total states (s_i,I^K) are examined and by using step 1, the corresponding conditions are listed in the table. That is, if state A is identified with state B then the condition denoted by (AB) is placed in the table. For all total states (s_i,I^K) each conditon of identification is listed in a separate row for each input I^K. As an example, consider state 1 and input I^1 from Table 4.5(a), i.e., situation 2 occurs. That is, one state (namely state 6) exists such

that $N(6,I^1)=N(1,I^1)$ and $O(6,I^1)=O(1,I^1)$. Therefore, (17) is placed in a row of state 1 under column π' of the condition table I^1, as shown in Table 4.5(b). Also, (76) is placed under column π' and (16) is placed under column π of table I^1, in a different row from (17). Likewise, the remaining entries of the two condition tables are obtained as given in Table 4.5(b). An identification table is then formed from the condition tables by merging similar columns of each condition table. Merging means that the same columns for each of the condition tables are combined. In other words, each row of one conditon table is combined with all the other rows in the corresponding state of the other condition tables. In each case the π' condition of one table are merged with the π' conditions of another table. Also, the π conditions of each table is merged. Note that the π' and π conditions are never combined. As an example, Table 4.5(c) gives the identification table for Table 4.5(b). This table is then minimized by applying the transitive law to each row. For example (17) and (14) are identified in the same row and column of Table 4.5(c), and by applying the transitive law the single condition (147) is formed. It should also be noted that when applying the transitive law the condition $\pi\cdot\pi'=0$ must also be preserved. If this can not be done then this row of the identification table is eliminated. As an example, consider state 2, the second set of conditions given is that (23)(35) is identified in π' and (25) is identified in π. Applying the

TABLE 4.5

(a) Next-state Table of Machine M
(b) Condition Table

	I^1	I^2
1	7, 00	4, 11
2	3, 00	3, 01
3	2, 01	6, 00
4	1, 11	4, 00
5	8, 11	3, 01
6	7, 00	7, 10
7	5, 10	6, 00
8	5, 10	4, 00

(a)

	I^1		I^2	
	π'	π	π'	π
1	(17) (76)	(16)	(14)	
2	(23)		(23) (35)	(25)
3	(23)		(36) (67)	(37)
4	(14)			
5	(58)		(53) (32)	(25)
6	(67) (17)	(16)	(67)	
7	(57) (58)	(78)	(76) (63)	(37)
8	(58) (57)	(78)	(48)	(48)

(b)

TABLE 4.5 (continued)
(c) Identification Table
(d) Minimized Identification Table

	π'	π
1	(17) (14) (76) (14)	(16)
2	(23) (23) (35)	(25)
3	(23) (36) (23) (67)	(37)
4	(14)	
5	(58) (53) (58) (32)	(25)
6	(67) (67) (17)	(16)
7	(75) (76) (58) (76) (75) (63) (58) (63)	(78) (37) (78) (37)
8	(58) (48) (57) (48) (58) (57)	(78) (48) (78) (48)

(c)

	π'	π
1	(147) (76) (14)	(16)
2	(23)	
3	(236) (23) (67)	(37)
4	(14)	
5	(358) (23) (58)	(25)
6	(67)	
7	(567) (58) (67) (36) (57) (36) (58)	(78) (37) (378)
8	(458) (57) (48) (58) (57)	(78) (48) (478)

(d)

transitive law would mean that (235) is contained in a block of π' and (25) is contained in a block of π. This condition cannot be used since $\pi \bullet \pi' = 0$ would result, and is discarded. The final minimized identification table is given in Table 4.5(d).

The identification table is then used to find a pair of partitions that satisfies Theorem 4.2. The following algorithm gives a method of finding such a pair from the minimized identification table:

Algorithm 4.2

1. Consider each state without optional rows (an optional row is a row with more than one condition row per state). An optional row which covers another optional row is removed from the identification table.
2. The states without optional rows are considered and a partition π'_j is formed by letting each condition for each of these states be a block of π'_j.
3. Those states with optional rows having an unspecified entry under π and an identity already covered by π' are discarded. The reason being that they do not place any further conditions on the pair (π', π).
4. Let the remaining states be $s_t, s_{t+1}, \ldots, s_{t+m}$. Let $P'_t, P'_{t+1}, \ldots, P'_{t+m}$ and $P_t, P_{t+}, \ldots, P_{t+m}$ be the subsets of partitions obtained by an arbitrary row selection

which refer to these states. The P'_t partitions are formed from the identities under the π' column, whereas the P_t partitions are formed from the identities under the π column. A pair of partitions (π'_t,π_t) can be determined as follows: $\pi'_t=\pi'_t+P'_t+P'_{t+1}+\ldots+P_{t+m}$ and $\pi_t=P_t+P_{t+1}+\ldots+P_{t+m}$. The set of pairs (π'_T,π_T) are obtained by all possible row selections. By row selection it is meant that one and only one row per state is chosen.

5. If a partition in the set of pairs (π'_T,π_T) is incompletely specified, the remaining states are added, preserving $\pi\cdot\pi'=0$.

If, after performing Algorithm 4.2 there is at least one pair in the set (π'_T,π_T), step 2 is executed.

Step 2: Select from the set of partitions given by Algorithm 4.2, those pairs that will satisfy condition 1 of Theorem 4.2. This simply consists of finding the pair of partitions (π'_t,π_t) with $\lceil\log_2 n(\pi')\rceil = h$ and $\lceil\log_2 n(\pi)\rceil = s-h$, where h is the largest.

The above method in achieving its goal of finding two partitions π and π, carries out two basic processes.

1. It places the states from a given next-state table into blocks of π and π' according to condition 2 of Theorem 4.2. In other words, step 1 finds only those partitions that can possibly satisfy condition 2 of the theorem.

2. The method then select the partitions π and π' that satisfy the conditions; $\pi \cdot \pi' = 0$, $\lceil \log_2 n(\pi') \rceil = h$ and $\lceil \log_2 n(\pi) \rceil = s-h$. That is, condition 1 of the theorem is satisfied.

Therefore, if two partitions π and π' can be found, this method assures that all the conditions of Theorem 4.2 are satisfied. If no partitions exist then no hybrid circuit exists for the given next-state table.

To illustrate this method, consider the identification Table 4.5(d). The states without optional rows are 2,4 and 6, therefore, $\pi_j = \{\overline{23}; \overline{14}; \overline{67}\}$. Consider the states with optional rows, i.e., states 1,3,5,7 and 8. As an example, one arbitrary row selection gives $P'_1 = \{\overline{14}; \overline{67}\}$, $P'_3 = \{\overline{23}; \overline{67}\}$, $P'_5 = \{\overline{23}; \overline{58}\}$, $P'_7 = \{\overline{67}; \overline{58}\}$ and $P'_8 = \{\overline{58}\}$. Thus, the partiton $\pi'_t = \pi'_j + P'_1 + P'_3 + P'_5 + P'_7 + P'_8 = \{\overline{23}; \overline{14}; \overline{67}; \overline{58}\}$. Likewise, for this row selection the partiton $\pi_t = \{\overline{16}\} + \{\overline{37}\} + \{\overline{25}\} + \{\overline{78}\} + \{\overline{48}\} = \{\overline{3478}; \overline{16}; \overline{25}\}$ can be obtained. To satisfy condition 1 of Theorem 4.1, it is possible to join the blocks $\overline{16}$ and $\overline{25}$, thus $\pi_t = \{\overline{3478}; \overline{1256}\}$. The pair of partitions (π'_t, π_t) obtained from this arbitrary row selection satisfies Theorem 4.2. A hybrid circuit can be built with the delay free internal variables assigned to the partition π'_t. Note, in order to find the partition that will give the maximum number of removable delays, the complete set of partition pairs (π'_T, π_T) must be obtained. The pair in which π'_t has the largest number of blocks is then used to realize the hybrid machine.

The above method assumes that the machine has p internal states such that $p = 2^s$; however, it is also possible to apply the method to a machine that has less than 2^s internal states. The state assignment given by this method realizes the machine with the least number of delay elements that can be removed. In other words, the number of removable delay elements represents a lower limit. It might be possible to increase this number by considering all possible selections of don't care entries. The problem of searching for the best specification of don't care entries for which the delays are fewest is a difficult problem. In some cases the extra states can be specified with entries identical to some other row of the original machine; however, the best choice is usually not obtained this way. Specifying don't care entries must be done in such a way that would allow some existing conditions to be split and still maintain the conditions stated in Theorem 4.2. The importance of carefully filling in don't care states is illustrated in [9].

The delay elements referred to in this chapter are simple delay elements. If the hybrid machines are to be realized with clocked flip-flops such as J-K or R-S, the method discussed earlier can still be used. However, when designing the circuit the process must be carried one step further. Once the transition table is formed it then has to be split into two separate tables. One is the transition

table of all the variables of the feedback paths from which the delay elements are removed. The other is the excitation table for the feedback paths from which the delay elements are not removed. The next-state equations are obtained from the transition table, and the flip-flop input equations are obtained from the excitation table. These two sets of equations are then used to realize the hybrid circuit.

4.2 Insertion of clocked Memory Elements

The object of this section is to develop a method for obtaining hybrid circuits from asynchronous circuits. The study deals with inserting clocked delay elements into the feedback paths of existing asynchronous circuits. A trivial solution to this problem merely inserts delays into all feedback paths. However, this may be undesirable since the circuit becomes completely synchronous. Therefore, what is desired is a method of inserting less delays than there are feedback paths.

The solution to this problem begins with considering asynchronous circuits. In asynchronous circuits two basic modes of operations are possible; normal and fundamental. Each mode will be considered separately since different modes produce different results in the operation of the hybrid machine. A fundamental mode flow table is one in which any transition leads directly to a stable state and no output is required to change more than once during a

transition. The inputs to a fundamental mode flow table are assumed to be stable and do not change until the circuit reaches a stable state.

With this in mind the following theorem gives the conditions needed to insert clocked delay elements into asynchronous machines operating in fundamental mode.

THEOREM 4.3 The clocked delay elements D_h, $j=1,\ldots,h$, can be inserted into the feedback paths $Y_1'-y_1,\ldots,Y_h'-y_h$ of an asynchronous circuit M without changing its behavior if and only if for any total state A, producing the output O(A), the associated hybrid transition state H[A] also produces the output O(A).

Proof: Let A be an input at any time t+kT, where T is the length of one clock pulse and let T' be the delay time through the combinational circuit, where $0<T'<T$. Then $t+kT+T' < t+(k+1)T$; That is, at time t+kT+T' the combinational circuit will see the associated hybrid transition state H[A]. When this occurs two cases are possible.

Case 1; If $y_j=Y_j'$ for all $j=h,\ldots,s$ then H[A]=A and the output state of the combinational circuit does not change, which implies the output is constant.

Case 2; If $y_j \neq Y_j'$ for at least one value of j, then $H[A] \neq A$. The combinational circuit will then see the new input, H[A], at time t+kT+T'. Thus, the output from time t+kT+T' to t+(k+1)T will be preserved if and only if

H[A] produces the same output as A. If and only if this holds for each input A then the insertion of h delays will never affect the external performance of M.

This theorem gives only the conditions needed to insert delays and does not give a way of doing it. A systematic approach is presented in the following algorithm and gives a method for inserting delays. It is assumed that the number, n, of delays to be inserted is less than s and can be arbitrarily chosen. If it is desirable to insert the maximum number and still have at least one feedback path that is delay free then n is initially set to s-1.

ALGORITHM 4.3

Let: K denote the collection of $\binom{s}{h}$ distinct combinations of integers 1,...,s,

k_i denote the elements of the set K, for $1<i<\binom{s}{h}$.

1. Set h=n.
2. Set h=h-1, find the set K for this h.
3. Set i=1
4. Calculate A, its associated hybrid transition state H[A] and O(A) as specified in Theorem 4.3 for feedback paths $Y_j^!-y_j$, $j=k_i$, and any transition.
5. If H[A] is equal to a don't care state, specify this state with the same next state and output of A and/or compare the output produced by H[A] to O(A). If they are not equal go to step 7.
6. If A, H[A] and O(A) are calculated for all total states go to step 9; otherwise, calculate A, H[A]

and O(A) for a new transition state and go to step 4.

7. Set i=i+1.

8. If $i>\binom{s}{h}$ go to step 11, otherwise go to step 4.

9. The h delay elements D_j, $j=k_i$, can be inserted and the process stops.

10. If h>1 then go to step 3.

11. No delay elements can be inserted, and the process stops.

The above discussion considered only fundamental mode flow tables; that is, tables in which an input change resulted in a single transition. It must be assumed that the input changes of a hybrid machine obtained in the above manner be at least one clock pulse apart. The reason is that if the input change is less than one clock pulse, the behavior of the machine might be changed. In this case the asynchronous variables are allowed to make more than one transition before the synchronous variables make their first transition. This could cause the circuit to operate incorrectly.

Circuits operating in normal mode are considered next. A machine operates in <u>normal mode</u> if and only if the inputs are never changed unless the circuit is in a stable state. Only those flow tables with more than one transition will be considered since tables with single transitions are accounted for by fundamental mode.

Consider an asynchronous flow table that has at least two transient states in a transition from one stable state to another. In other words, for a single input change a series of internal state transitions exist. Let t be the maximum time it takes the asynchronous circuit to make one transition, i.e., t is the delay time through the combinational circuit for one change of the total state. This point is illustrated in Table 4.6; t is the time it takes the circuit to reach state B, after an input change, if the system was in state A with an input of 1. The arrow on the state table indicates the transition from A to B made in time t. Assuming that the delay time through the circuit is constant, then the total transition time for the circuit to change from state A to C is 2t. From the definition of normal mode it follows that the interval 2t must be less than the minimum time that the external inputs change. It should be noted that t is not constant, but for the purpose of this thesis it will be assumed that t is within some limits, i.e., $t_0-\xi<t<t_0+\xi$.

Let the flow table of Table 4.6 be realized by a hybrid machine and consider the problems of such a realization. Assume the hybrid machine operates correctly, then the external behavior is the same as the external behavior of an asynchronous machine realizing this flow table. The hybrid machine will pass through the transient state when a transition from one stable state to another is made. The

y-variables that have no delays in their feedback paths are allowed to change. Therefore, the machine will either stay in its original state or change to its associated hybrid transition state. In either case the circuit will be stable for a fixed amount of time, which is one clock pulse minus the delay time through the combinational circuit. By the definition of normal mode the input could now change after 2t units of time. The inputs might change in the hybrid machine before the first or second transition since t is very small compared to one clock pulse. This result would cause the behavior of the hybrid circuit to be different than that of the asynchronous circuit realizing the same flow table. In order for the hybrid circuit to operate correctly, the restriction is imposed that the inputs must be separated by at least two clock pulses.

TABLE 4.6

Asynchronous next state table

	0	1
A	B	(A)
B	C	(B)
C	(C)	B
D	B	(D)
E	C	(E)

It can now be seen that Theorem 4.3 can be used to

realize an asynchronous circuit operating in normal mode by a hybrid circuit for this specific case. In general, a machine operating in normal mode having multiple transitions can still be realized by a hybrid machine. To satisfy Theorem 4.3 the inputs must be separated by at least n clock pulses, where n is the number of transitions of the path containing the maximum number of transitions.

According to McCluskey [23], a machine operating in normal mode, after all input changes, must reach either a stable condition or a cycle. A cycle is a succession of internal states repeated indefinitly. For a flow table that contains a cycle two possiblities exist: One is when the next state can be uniquely determined and the other when the next-state can not be uniquely determined. To illustrate the two possibilities consider the two flow tables in Table 4.7. Let the system realizing Table 4.7(a) have an input of I^2 and be in state 1 or 2. For this case the system will be alternating between state 1 and 2. When the input is changed from I^2 to I^1, no matter what state of the cycle the system is in, the next state will always be 3. However, consider Table 4.7(b) in the same situation, when the input is changed to I^1 the next state can not be directly determined by the flow table. If the system was in state 1 when the input was changed it will remain in state 1. If the system was in state 2 the next state would be 3.

TABLE 4.7

Asynchronous next-state table with cycles

	I^1	I^2
1	3	2
2	3	1
3	(3)	2

(a)

	I^1	I^2
1	(1)	2
2	3	1
3	(3)	2

(b)

The first possibility discussed is the case where the next state can be uniquely determined even if it is not known what state of the cycle the circuit is in. The second case can be considered rare and the use of such a flow table is not clear.

Theorem 4.3 can still be used to find a hybrid machine that will realize this type of flow table; however, a restriction must be placed on the inputs. Let n be the number of transitions in the path that contains the maximum number of state changes from one stable state to another. To determine n, the cycles are considered as stable states. The inputs are again restricted to change at the earliest n clock pulses apart.

The above discussion implies that all inputs must be separated by n clock pulses, but this need not be the case. If the n transitions occur in only one column, say I^n, then the only restriction that must be made is that the change

from I^n to I^x is n clock pulses after the change from I^z to I^z. In general, the change in inputs from I^n to I^x must be separated by at least the same number of clock pulses as there are transitions in the input column I^n.

Pulse outputs in the transient states of a machine operating in fundamental mode could create a problem if realized by a hybrid machine. One reason for having transient states and/or cycles is to give pulse outputs, but when such a table is realized by a hybrid circuit the length of the pulse will be affected. The transient output will remain on longer than it would for the asynchronous case. It must be assured that the change in the length of the pulse will not affect other machines. For example if such a circuit is to operate in series with some other machine, in which the outputs of the hybrid machine are used as inputs to this other machine, the behavior of the composite machine might be affected. It is, therefore, important to consider the outside environment of an asynchronous circuit before trying to realize it with a hybrid circuit.

4.3 Summary

This chapter has presented the structure and behavior of hybrid sequential machines. An existing theorem is described enabling a hybrid circuit to be obtained from a sycnhronous circuit by removing delay elements. To complement this, a theorem was presented for inserting delay

elements into asynchronous circuits to form a hybrid circuit. These results will be used in the next chapter to eliminate critical races and essential hazards in asynchronous circuits.

CHAPTER 5

RACES, HAZARDS and HYBRID CIRCUITS

The problem of races in hybrid machines is examined in this chapter. A method of resolving races in hybrid circuits will be explained so that the circuits will operate correctly. Also, some possible uses of hybrid machines will be considered, in particular for the elimination of both critical races and essential hazards from asynchronous circuits.

5.1 Races

A transition during which more than one internal variable changes is called a race. If the correct behavior of the circuit depends upon which variable changes first, i.e., the outcome of the race, then it is considered to be a critical race [15]. A race is noncritical if, regardless of the outcome of the race, the behavior of the circuit still conforms to that specified by the flow table [15,23].

Race conditions do not occur in synchronous circuits because the internal variables change simultaneously. On

the other hand race conditions do occur in asynchronous circuits and require careful analysis. In order to obtain reliable circuit operation, critical races must be eliminated. Three different approaches can be used to avoid critical races. One such approach is by directing the circuit through intermediate unstable states [18]. Another method is by selecting a state assignment which is race free [15,23]. The method of inserting asynchronous delays into one or more of the feedback paths is also a common method of avoiding critical races[10,23]. With this method, delays are inserted in such a way as to allow one of the racing variables to change before the other.

Each of the above methods has its disadvantages, i.e., the first and second approaches could lead to larger state assignments, while in the third method there may be more than one race with the same y-variables. If this is the case one y-variable might be required to win the race one time and under another circumstance it might be required to lose the race. The best method to use depends on the situation, i.e., no one method is always better than another. Quite often a combination of these methods are required. A new approach for resolving races is presented in Section 5.1.2.

One of the major problems that could arise in the conversion of a synchronous circuit to a hybrid one is that of creating races. In order for a hybrid circuit to operate

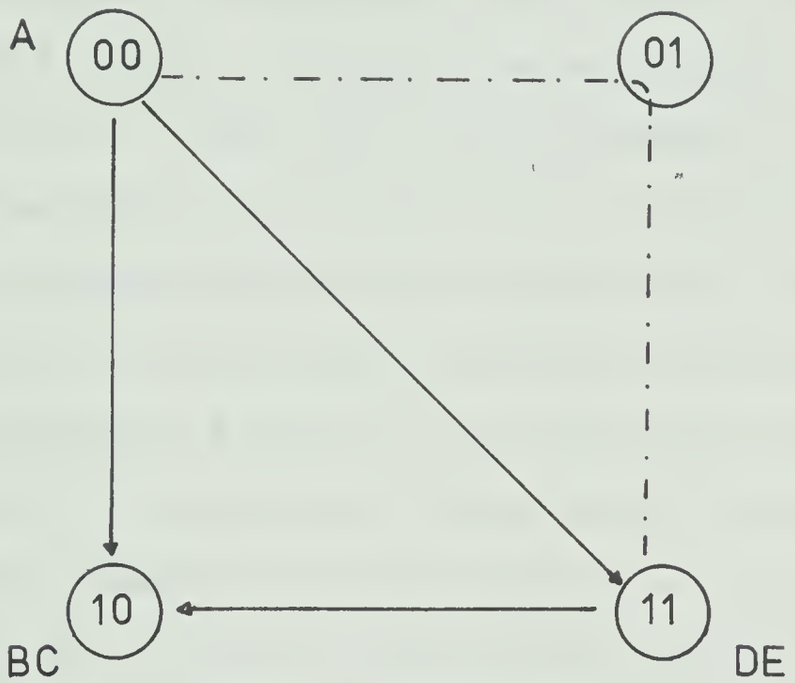


Figure 5.1

2-Cube for Table 5.1

with the same behavior as the original synchronous circuit all critical races must be eliminated. This problem of races in hybrid machines and how they can be resolved is dealt with in the next section.

5.1.1. Races In Hybrid Circuits

In order to illustrate the existence of critical races in hybrid circuits consider the synchronous flow table given in Table 5.1. Using the method discussed in Section 4.1, this flow table can be realized by a hybrid machine if the clocked memory elements are removed from feedback paths of y_2 and y_3. Let the hybrid machine, realizing Table 5.1 at some particular time kt, be in state 000 with input 0. The next state of the hybrid system should be 011. This situation results in a race condition since N(000,0)=011 and both y_2 and y_3 are to change simultaneously. Now consider what would happen if y_2 changes before y_3. Under this condition the next-state of the system would be 010 and with the next clock pulse the system would then go to 110 which is different from the required next-state of 111. Thus, critical races can exist in the hybrid machine.

TABLE 5.1

Hybrid transition table to illustrate races

				0	1	0	1
A	0	0	0	011	010	0	1
B	0	1	0	110	010	1	1
C	1	1	0	010	010	1	1
D	1	1	1	010	011	1	0
E	0	1	1	111	011	0	0
				Y'_1	Y'_2	Y'_3	z

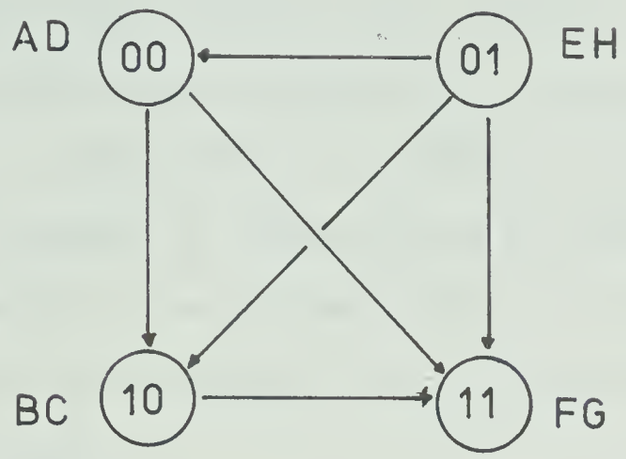
Consider what would happen if the standard techniques for eliminating races are applied individually for this machine. The first technique of directing the circuit through unstable intermediate states is inadequate, since there exists no state that leads to 111 other than 011. Finding a new state assignment may be unsatisfactory, since it might not be possible to realize the new transition table by a hybrid circuit. In addition, changing the state assignment might result in the removal of different delay elements and thus, new races might be introduced.

The last technique of resolving the race by inserting asynchronous delay elements into one of the feedback paths seems to be the best solution; however, this technique also has its disadvantages. For example, if y_3 was delayed to enable y_2 to change first, the system would go from state

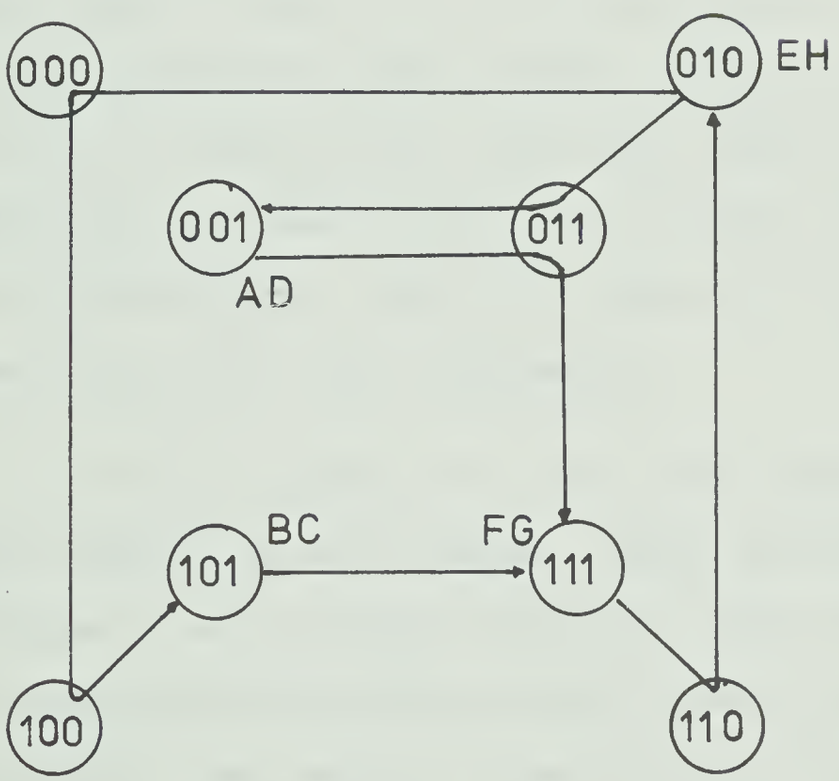
000 to 010, which is still the wrong next-state. Likewise, if y_2 was delayed a similar situation would occur. Thus, no individual method can be applied to this machine. However, a combination of techniques which can be used to resolve races in hybrid circuits and is now discussed.

The above discussion considered both the synchronous and asynchronous variables of the hybrid machine. Recall that the non-delayed internal variables of a hybrid machine are called asynchronous and the delayed ones synchronous. It will be assumed that no races exist between different types of variables. This assumption can be justified, since the clock pulse can be delayed so that the asynchronous variables will reach a stable state before the synchronous variables change. It should be noted that the time it takes the internal variables to change is increased. Consequently, the problem of races is confined to the asynchronous variables because these are not required to change simultaneously. By considering only the asynchronous variables it is possible to resolve races in the hybrid machine. Races can be resolved by using transient states and modifying the existing state assignment. However, it must be ensured that the race free assignment can still realize the hybrid circuit.

To discuss the race problem in hybrid machines consider Table 5.1, which is realizable with a hybrid circuit. The delay elements are removed from the feedback paths y_2 and y_3



(a)



(b)

Figure 5.2

(a) 2-Cube for Table 5.3

(b) 3 Cube for Table 5.3

and are checked to ascertain if races exist in the circuit.

Using the notation that states A, BC, and DE represent the asynchronous variables 00, 10 and 11 respectively, a 2-cube is constructed, as given in Fig. 5.1. This 2-cube is used to detect if there are any races. It is obvious that there exists a race from state 1 to state 3. Furthermore, this race is a critical race, since state 2 is stable under input column 0.

To resolve the critical race, the system is made to pass through the transient state 01, as indicated by the dotted line in Fig. 5.1. The next state of 000, in the original transition table is then changed to 001, and the state 001 is added to the table and N(001,0)=011. No other alterations are made to Table 5.1, since there existed only one race condition. Table 5.2, gives the new transition table with a race free assignment.

It should be noted that the state assignment of the hybrid machine is not changed. Also, the synchronous variables of any created transition state are the same as the synchronous variables of the initial state of the transition. The output of the transient state is given the same output as the initial state of the transition. Thus, if the transient state satisfies Theorem 4.1, the new transition table can still be realized by a hybrid circuit.

TABLE 5.2

Hybrid machine with race free assignment

				0	1	0	1
A	0	0	0	011	010	0	1
B	0	1	0	110	010	1	1
C	1	1	0	010	010	1	1
D	1	1	1	010	011	1	0
E	0	1	1	|||	011	0	0
F	0	0	1	111	---	0	-
				Y'_1	Y'_2	Y'_3	z

To show that the new transition table can, in fact, be realized by a hybrid circuit it must be shown that Theorem 4.1 is satisfied for each transient state. To show this, let $C=y'_1,\ldots,y'_s,x'_1,\ldots,x'_n$ be a transient state giving an output of $O[C]$, and let the associated hybrid transition state of C be $H[C]=Y''_1,\ldots,Y''_h,y'_{h+1},\ldots,y'_s,x'_1,\ldots,x'_n$. According to Theorem 4.1 $H[C]$ must give the same output $O[C]$. Let $A=y_1,\ldots,y_s,x_1,\ldots,x_n$ be the intitial state in the transition, giving the output $O[A]=z_1,\ldots,z_m,Y'_1,\ldots,Y'_s$. Let the associated hybrid transition state of A be $H[A]=Y'_1,\ldots,Y'_h,y_{h+1},\ldots,y_s,x_1,\ldots,x_n$. The synchronous variables of C are equal to the synchronous variables of A, since they were not changed. The input of A and C are the same since the transition remains in the same input column. Thus $C=y'_1,\ldots,y'_h,y_{h+1},\ldots,y_s,x_1,\ldots,x_n$ and $H[C]=Y''_1,\ldots,Y''_h,$

$y_{h+1},\ldots, y_s, x_1,\ldots,x_n$. Let the original $N(A,I) = Y'_1,\ldots, Y'_h, y_{h+1},\ldots,y_s$ but since C is a transient state $N(C,I) = N(A,I)$, the next-state asynchronous variables of C are equal to that of A. This means that $Y''_1=Y'_1, \ldots, Y''_h=Y'_h$, making $H[C] = Y'_1,\ldots,Y'_h,y_{h+1},\ldots,y_s,x_1,\ldots,x_n$ which implies $H[C]=H[A]$.

H[C] gives the same output as C for the following reasons. C is the transient state of A, therefore has the same output as A. Also, H[A] has the same output as A since it is the hybrid transition state associated with A and already satisfies the theorem. With these two statements and the fact that $H[C]=H[A]$, implies that the output of H[C] is the same as the output of C. Thus, Theorem 4.1 is satisfied for this transient state.

To explain why the behavior is not changed when the next-state of A is changed to the transient state, consider the following argument. The synchronous variables of the transient state C are the same as the synchronous variables of A, therefore, the circuit makes the transition from state A to C. The transition is then made from C to H[C], since H[C] has the same synchronous variables as C. Therefore, the transition is still to H[A], since $H[A]=H[C]$. With the next clock pulse H[A] is then allowed to make the transition to N(A), as was required. Furthermore, all transient states created by this method will satisfy the theorem i.e., the above argument can be applied to all transient states.

Therefore, the transition table formed by adding intermediate unstable states can be realized by a hybrid circuit.

The above method for eliminating critical races is satisfactory if there exist unused intermediate states. If no intermediate states are present, they can be created by adding extra internal variables to the assignment. However, care must be taken to ensure that the internal variables already assigned to each state are not changed. The following example will illustrate how extra internal variables can be added to obtain a race free assignment.

Consider the hybrid machine, whose transition Table is given in Table 5.3. The 2-cube, Fig. 5.2(a), of the asynchronous variables is used to determine if there are any races. It can be clearly seen from Fig. 5.2(a), that races exist in this hybrid machine; furthermore, these races cannot be resolved by just a two variable assignment. An assignment with more than two variables is then required to resolve the race. The internal variables already assigned are not changed since all hybrid machines are dependent on the assignment. Thus, the asynchronous variables of state AD must be coded as either 000 or 001.

The 3-cube, Fig. 5.2(b), is now used to find a race free assignment. The codes 001, 101, 111, and 010 are assigned to the states AD, BC, FG and EH respectively. The

TABLE 5.3

Hybrid machine with races

				0	1	0	1
A	0	0	0	111	100	00	11
B	0	1	0	110	110	00	01
C	1	1	0	010	011	01	00
D	1	0	0	000	100	11	00
E	0	0	1	101	110	11	01
F	0	1	1	111	111	00	10
G	1	1	1	001	011	10	00
H	1	0	1	001	100	10	00
				$Y_1' Y_2' Y_3'$		$z_1 z_2$	

race free assignment is obtained from the 3-cube where the arrows represent the required transitions, as given in Fig. 5.2(c). Note that the synchronous variables of each state are not changed; furthermore, if a transition has an unstable intermediate state, then the synchronous variables of the transient state must be the same as the initial state of the transition. The resulting race free assignment is given in Table 5.4.

Using the same argument given previously it can be shown that each transient state created by this method will satisfy Theorem 4.1. In other words, the new transition table is still realizable by a hybrid circuit and is free of races.

TABLE 5.4

Hybrid machine with race free assignment

		0	1	0	1
A	0 0 0 1	0011	1001	00	11
B	0 1 0 1	1101	1101	00	01
C	1 1 0 1	0101	0111	01	00
D	1 0 0 1	0001	1001	11	00
E	0 0 1 0	1010	0000	11	01
F	0 1 1 1	1111	1111	00	10
G	1 1 1 1	0110	0111	00	10
H	1 0 1 0	0010	1011	10	00
I	0 0 1 1	1111	----	00	--
J	0 0 0 0	----	0100	--	11
K	0 1 0 0	----	1101	--	11
L	0 1 1 0	1010	----	10	--
M	1 0 1 1	----	1001	--	00
		Y'_1 Y'_2 Y'_3 Y'_4		z_1	z_2

5.1.2 Race Elimination by Hybrid Machines

This section will investigate the possibility of eliminating critical races from asynchronous circuits by inserting clocked delay elelments. Trivially, all races can be resolved by inserting clocked delays into all feedback paths, since all asynchronous machines can be realized by a synchronous circuit. A more desirable approach would be to

have clocked delays inserted only into the feedback paths of those variables that are involved in the race. Inserting delays in this fashion would undoubtedly resolve all races; however, the behavior of the machine could be changed. To illustrate this, consider the circuit which is defined by Table 5.5. Clearly, a race condition exists between the two internal variables. If a clocked delay element is inserted into one of the feedback paths, say y_2, then the behavior of the machine is changed. As an example let the circuit be in state 00 with an input of 1. When the input is changed to 0 the asynchronous variable is allowed to make its change, i.e., the machine goes to state 01. In the event of the next clock pulse the synchronous variable is allowed to make its change, and the final state reached is 01. Therefore, the circuit reaches the state 01 which is different from the state 11 as specified by the table. Consequently, the behavior of the machine can be easily changed by inserting delay elements. However, recollect that it is possible to convert some asynchronous machines to hybrid machines without changing its state behavior [Section 4.2].

Specifically, Algorithm 4.3 was used to give the feedback paths of the variables that could be delayed. Recall from Chapter 4 that Algorithm 4.3 describes a method of finding the feedback paths into which delays can be inserted. In this Algorithm, n was the maximum number of delays that were required to be inserted and K the set of all combinations of internal variables. It is possible to

use Algorithm 4.3 to determine whether the insertion of clocked delay elements will eliminate the races. In order to do this, n is set to the number of racing internal variables minus one and the set K changed so that it contains combinations of all those variables that are involved in races.

TABLE 5.5

Asynchronous transition table with races

$y_1 y_2$	0	1
0 0	11	00
0 1	01	00
1 1	11	10
1 0	01	10

$Y_1' \; Y_2'$

The application of Algorithm 4.3 will result in one of two possible outcomes. One is that no clocked delay elements can be inserted into any of the feedback paths. In this case no race can be resolved by using a hybrid machine. The other outcome is that delay elements can be inserted into all or some of the feedback paths of the variables that are involved in a race. If all the feedback paths of the racing variables contain clocked delays, then trivially all races of the asynchronous machine can be resolved by a

hybrid circuit. If, however, delay elements can only be inserted into some of the feedback paths, an additional check must be made to ascertain whether all critical races have been removed. This can be done by considering those feedback paths that do not contain clocked delays.

If not enough delay elements may be inserted to resolve all races, the following can be done. Apply the technique of resolving races in hybrid machines [Section 5.1.1]. In other words some of the races would be resolved by adding clocked delay elements. The remaining races would be resolved by modifying the existing state assignment and by using transient states.

The hybrid machine will resolve critical races for the following reason. The asynchronous variables of the hybrid circuit that are not racing and one of the variables that was racing, are allowed to make their transition first. The remaining variables will now make their transition simultaneously; hence, no races can result.

To illustrate this procedure consider the asynchronous transition table given in Table 5.6. It can be seen from the table that two races exist. One race occurs when the circuit makes a transition from state 000 to state 011 under input 0. In this case the y_2 and y_3 internal variables are racing. The other race occurs when the system changes from state 100 to 011 under input 0,; consequently, the variables y_1,y_2 and y_3 are racing. The set K will be {12, 13, 23} and

n is set equal to two, since the number of variables involved in the race is three. Algorithm 4.3 yields the feedback paths into which clocked delay elements can be inserted as Y_2-y_2 and Y_3-y_3. Thus, the races that would have occurred in an asynchronous circuit with these variables have been resolved since y_2 and y_3 change simultaneously and no race results. Also, if y_1 is allowed to change first, hence no race exists between y_1 and y_2 or y_3.

TABLE 5.6

Asynchronous circuit with races

	0	1
0 0 0	011	000
0 0 1	101	001
0 1 0	010	001
0 1 1	011	010
1 0 0	011	100
1 0 1	101	001
1 1 0	010	110
1 1 1	101	111

Y_1' Y_2' Y_3'

From the above discussion it is seen that the possibility of resolving critical races by using a hybrid

circuit does exist. However, this process is limited only to asynchronous machines that satisfy Theorem 4.3

The advantage of using hybrid machines to resolve races is that the procedure is simple and straight-forward, in fact, it could be easily programmed. Implementing this procedure would make it an acceptable method for large machines, since resolving races in machines with more than four variables is very laborious. Another advantage is that no calculations for the length of each delay is needed, as is the case when asynchronous delays are to be inserted.

In short, using hybrid machines does not present a complete method of resolving races, but rather gives an alternate approach from those methods already developed.

5.2 Hazards

Electronic networks differ from theoretical networks mainly due to delays in the circuit. That is, signals are never transmitted instantaneously through the circuit nor do gates react in zero time. Delays in the circuit may cause it to operate incorrectly and such a circuit is said to contain a hazard [32].

Two types of hazards are possible in sequential circuits. A transient hazard is present if momentary false output signals occur at one or more output terminals following certain changes in the input state. A steady-

state hazard will exist if a circuit can enter the wrong internal state after certain input changes. One type of steady-state hazard is the essential hazard. A sequential circuit contains an essential hazard if for some initial state and input variable x, three consecutive changes in x take the system to a state that is different from, and not equivalent to, the state reached after a single change in x.

Transient hazards exist in combinational circuits and [16] has shown that they can always be avoided by appropriate design techniques. It is assumed that all combinational circuits will be designed free of transient hazards. There are different types of steady-state hazards such as the critical race condition. However, this source of hazards has been dealt with in Section 5.1.1 and will not be considered here. The class of steady-state hazards that will be studied in this section is the essential hazard.

To illustrate an essential hazard consider the state table, given in Table 5.7. Let the system be in state A under input of 0. When the input changes from 0 to 1, y_2 becomes unstable and switches to a 1. If the effect of the y_2 action penetrates to y_1 before the direct effect of the change in x then as far as y_1 is concerned, the system is in state B, where y_1 is unstable. This causes y_1 to switch to 1 and the system now moves to state C. Once this occurs, y_1 remains at 1 even after the effect of the input change reaches it. Finally, since y_2 is unstable in state C, with

input 1, it changes again and the system moves to state D, where it remains. This causes the system to be in the wrong state which results in the incorrect operation of the circuit.

TABLE 5.7

Asynchronous next-state table with essential hazard

$y_1 y_2$		0	1
0 0	A	A	B
0 1	B	C	B
1 0	C	C	D
1 1	D	A	D

Solutions to the essential hazard problem involve inserting delays into one or more of the feedback paths [16,21,27,32]. These techniques all involve the use of asynchronous delay devices where the length of the delay must be calculated. This section will deal exclusively with the use of hybrid machines to eliminate essential hazards by inserting clocked delay elements into asynchronous circuits.

Before approaching the problem, the following assumptions will be made:

1. Transient hazards are not present in the network.
2. All circuits operate in normal mode.
3. There are no races in the internal variables.

5.2.1 Essential hazards

Unger [32] has shown that if a state table contains an essential hazard then any proper circuit realization must contain at least one delay element. The delays are asynchronous and each value must be calculated, within limits, in order for the circuit to operate properly. It will now be shown, that clocked delays can be inserted into asynchronous circuits to resolve essential hazards.

In Unger's method [32] for essential hazard correction the condition is made that there is no upper bound for the value of stray delays. However, Lerner [21] introduced the idea of upper bounds for the magnitude of stray delays. He explained that having no upper bounds was a theoretical problem. A more realistic approach, for all practical aspects, would be to have upper bounds. A general review of the two methods is found in [14]. In order to correct essential hazards by using a hybrid machine the time of each clock pulse must be set to the upper bound on stay delays.

To illustrate essential hazard correction using hybrid circuits, consider the transition table, Table 5.8,. This asynchronous machine has its state behavior described by Table 5.6 and, as explained earlier, contains an essential hazard. Applying Theorem 4.3 to Table 5.8, it is seen that no clocked delay can be inserted into feedback path Y_1'-y_1, but that a clocked delay may be inserted into the Y_2'-y_2 feedback path. With a clock delay element in the Y_2'-y_2

feedback path there no longer exists an essential hazard for the following reasons: Inserting a clocked delay into the $Y_2' - y_2$ feedback path allows the y_1 variable to change first. In other words, there is no way the y_2 action can penetrate to y_1 before the direct effect of the input is felt by y_2. This action then resolves the essential hazard that was present in the circuit.

TABLE 5.8

Asynchronous transition table with essential hazard

$y_1 y_2$	0	1
0 0	00	01
0 1	10	01
1 0	10	11
1 1	00	11

It follows, from this example, that hybrid machines can be used to resolve essential hazards. Obviously not all essential hazards can be resolved by using hybrid machines, since not all asynchronous state tables can be realized with a hybrid circuit.

Unlike the critical race problem, essential hazards are not dependent on the state assignment, whereas hybrid machines are. Thus, if one state assignment does not lead to a solution it might be possible to change the assignment

so that a hybrid circuit might be found that will resolve the hazard. There exists no convenient method of finding the best state assignment that will eliminate hazards. The hybrid circuit does not offer a complete solution of eliminating essential hazards; however, whenever they can be used they have the advantage that the delay time does not have to be calculated.

5.3 Summary

This chapter has endeavored to look at one of the problems that exists when a synchronous machine is realized by a hybrid circuit. In particular, a solution was given that would resolve any race in a hybrid circuit. The discussion presented also outlines two schemes in which hybrid machines could be utilized, in particular for resolving races and hazards. The approach described does not claim to make existent methods obsolete, but is intended as an alternate solution to the same old problems.

CHAPTER 6

Conclusions

Sequential machines have been divided into two seperate classes of machines, synchronous and asynchronous. Most networks are usually constructed of components of either one class or the other. Very rarely are machines built of both synchronous and asynchronous components. It was the intent of this research to explore the relationship of the two classes of machines so that networks could be built using components of both types. Conditions were given in which two different types of machines could be connected in series or in parallel. Although this study was limited to combining only two machines, the results obtained can be easily extended to connections of any number of machines.

The relationship of machines was further studied by examining serial and parallel decomposition. Of specific interest, was the study of decomposition in which the outcome resulted in one synchronous and one asynchronous submachine. The main purpose was to obtain a better understanding of the interconnection of synchronous and

asynchronous circuits. It also gave an insight into the role delay elements play in sequential machines. It was found that synchronous decomposition of this type decreases the number of delay elements which store the internal states. Whereas, in asynchronous decomposition the number of delay elements are increased.

With the development of the hybrid machine the two classes are brought even closer together. Hybrid machines can be obtained by either removing delay elements from synchronous circuits or by inserting delay elements into asynchronous circuits. A machine of this type possesses properties of both synchronous and asynchronous machines. It should be noted that there is a cost of building a hybrid machine from either type of machine. In other words, if hybrid circuits are obtained from synchronous ones, there exists the possibility that races are introduced into the circuit. On the other hand, if hybrid circuits were obtained from asynchronous circuits it is done at the expense of time, i.e., the hybrid circuit will not operate as fast as the equivalent asynchronous circuit.

Most modern computers are synchronous because they avoid such problems as hazards and races that may occur in asynchronous circuits. Since asynchronous machines are faster than synchronous ones, the investigation of hazard and race free realizations of asynchronous circuits is of interest. The problem of resolving races and hazards by

hybrid circuits is of importance because properties of both asynchronous and synchronous machines are employed.

The results of this investigation suggest several areas worthy of further investigation:

1. Apply the results obtained in this thesis to circuits containing more than one delay element per feedback path. One approach to this problem could be to consider feedback shift registers and apply the techniques presented.

2. Another idea worth exploring is that of finding partitions from asynchronous machines and relating them to hybrid machines. That is, obtaining partitions from asynchronous next-state tables such that an assignment acquired from these partitions can be used to realize a hybrid circuit. This could be done by expanding the work presented on decomposition of asynchronous machines.

3. Closely related to the above problem is that of finding an assignment that would realize a hybrid circuit containing no races.

It is believed that the ideas presented here offer a new approach for the further investigation of hybrid circuits and their application to asynchronous networks.

BIBLIOGRAPHY

1. Armstrong, D.B., Friedman, A.D. and Menon, P.R., "Sequential Circuits Without Inserted Delay Elements", I.E.E.E. Transactions on Computers, Vol. C-17, pp. 129-134, February, 1968.

2. Bartee, T.C., Lebow, I.L. and Reed, I.S., Theory and Design of Digital Machines, McGraw-Hill Book Company, New York, 1962.

3. Caldwell S.H., Switching Circuit and Logic Design, John Wiley and Sons, New York, 1958.

4. Friedman, A.D., "Feedback in Asynchronous Sequential Circuits", I.E.E.E. Transactions on Computers, Vol. EC-15, pp. 740-749, October, 1966.

5. Friedman, A.D., "Universal Single Transition Time Asynchronous State Assignments", I.E.E.E. Transactions on Computers, Vol. C-18, pp. 541-547, June, 1969.

6. Gerace, G.B. and Gesrti, G., "State Assignments For Reducing The Number Of Delay Elements In Sequential Machines", Information and Control, Vol. 10, pp. 223-253, 1967.

7. Gerace, G.B. and Gestri, G., "Decomposition of a Synchronous Sequential Machine into Synchronous and Asynchronous Submachines", Information and Control, Vol. 11, pp. 568-591, November-December, 1967.

8. Gerace, G.B. and Gestri, G., "Decomposition of a Synchronous Machine into an Asynchronous Submachine Driving a Synchronous One", Information and Control, Vol. 12, pp. 538-548, May-June, 1968.

9. Gerace, G.B. and Gestri, G. "Sequential Machines With Less Delay Elements than Feedback Paths", I.E.E.E. Transactions on Computers, Vol. C-18, pp. 132-144, February, 1969.

10. Givone, D.D., Introduction to Switching Circuit Theory, McGraw-Hill, New York, 1970.

11. Harlow, C. and Coates, C.L., "On the Structure of Realization Using Flip-Flop Memory Elements", Information and Control, Vol. 10, pp. 159-174, February, 1967.

12. Hartmanis, J. and Stearns, R.E., "Some Dangers in State Reduction of Sequential Machines", Information and Control, Vol. 5, pp. 252-260, 1962.

13. Hartmanis, J. and Stearns, R.E., Algebraic Structure of Sequential Machines, Prentice-Hall, Englewood Cliffs, N.J., 1966.

14. Hlavicka, j., "A Note on Essential Hazards", Seminarbericht Institute Fuer Theorie der Automater und Schaltnetzwerke NO. 15, Bonn, June 1969.

15. Huffman, D.A., "A Study of the Memory Requirements of Sequential Switching Circuits", Technical Report No. 293, M.I.T. Research Laboratory of Electronics, 1955.

16. Huffman, D.A., "Design of Hazard-Free Switching Circuits", Journal of Association for Computing Machinery, Vol. 4, pp. 47-62, January, 1957.

17. Kohavi, Z., Switching and Finite Automata Theory, McGraw - Hill, New York, 1970.

18. Krieger, M., Basic Switching Circuit Theory, Collier - MacMillan Limited, London, 1969.

19. Langdon, O.C., "Delay-Free Asynchronous Circuits With Contained Line Delays", I.E.E.E. Transactions on Computers, Vol. C-18, pp. 175-181, February, 1968.

20. Langdon, O.C., "Analysis of Asynchronous Circuits Under Different Delay Assumptions", I.E.E.E. Transactions on Computers, Vol. C-17, pp. 1131-1143, December, 1968.

21. Lerner, S.B., "Hazard Correction in Asynchronous Sequential Circuits", I.E.E.E. Transactions on Electronic Computers, Vol. EC-14, pp. 265-266, April, 1965.

22. Maki, G.K. and Tracey, J.H., "State Assignment Selection in Asynchronous Sequential Circuits", I.E.E.E. Transactions on Computers, Vol. C-19, pp.641-648, July, 1970.

23. McCluskey, E.J., "Fundamental Mode and Pulse Mode Sequential Circuits", Proceedings of IFIP Congress 1962, Vol. 2, pp. 725-730, 1962.

24. McCluskey, E.J., "Reduction of Feedback Loops in Sequential Circuits and Carry Leads in Iterative Networks", Information and Control, Vol. 6, pp. 99-118, 1963.

25. McCluskey,E.J., Introduction to Theory of Switching Circuits. McGraw-Hill, New York, 1965.

26. Morris, R.L. and Miller, J.R.(eds), Designing with TTL Integrated Circuits, McGraw-Hill, New York, 1971.

27. Meisel, W.S. and Kashef, R.S., "Hazards In Asynchronous Sequential Circuits", I.E.E.E. Transactions on Computers, Vol. C-18, pp. 752-759, August, 1969.

28. Miller, R.E., "Sequential Circuits and Machines", Switching Theory, Vol. 2, John Wiley and Sons, New York, 1965.

29. Morris, R.L. and Millar, I.R., Design with TTL Intregrated Circuits, McGraw-Hill, New York, 1971.

30. Tan, C., Menon, P.R. and Freidman, A.D., "Structural Simplification and Decomposition of Asynchronous Sequential Circuits", I.E.E.E. Transactions on Computers, Vol. C-18, pp. 830-838, September, 1969.

31. Tracey, J.H., "Internal State Assignments for Asynchronous Sequential Machines", I.E.E.E. Transaction on Electronic Computers, Vol. EC-15, pp. 551-560, August, 1966.

32. Unger, S.H., "Hazards and Delays in Asynchronous Sequential Switching Circuits", IRE Transactions on Circuit Theory, Vol. CT-6, pp. 12-25, March, 1959.

33. Unger, S.H., Asynchronous Sequential Switching Circuits, John Wiley and Sons, New York, 1969.

34. Wickes, W.E., Logic Design with Integrated Circuits, New York, 1968.